FACHHOCHSCHULE WEDEL

Bachelor-Thesis

im Fachbereich Medieninformatik

Thema:

Konvertierung der Darstellung von virtuellen Welten zu virtuellen Präsenzen

am Beispiel von Avataren in weblin und Second Life

Geprüft durch Prof. Dr. Christian-A. Bohn
Betreut durch Dr. Heiner Wolf

Alexander Treptow
21. Februar 2007

Alexander Treptow
Tannenallee 43
22844 Norderstedt
Tel. (0 40) 522 57 51
alextreptow@gmx.de

Prof. Dr. Christian-A. Bohn
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
Tel. (0 41 03) 80 48-40
bo@fh-wedel.de

Dr. Heiner Wolf
Zweitgeist GmbH
Stahltwiete 23
22761 Hamburg
Tel. (0 40) 374 236 090
wolf@zweitgeist.com

Eidesstattliche Erklärung

Ich, Alexander Treptow (Matrikelnummer 2622), erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Eidesstattliche Erklärung ... **II**
Inhaltsverzeichnis ... **III**
Abbildungsverzeichnis ... **V**
Tabellenverzeichnis ... **V**

1. Einleitung ... **1**
1.1. Virtuelle Welten ... **1**
1.2. Virtuelle Präsenzen ... **2**
1.3. weblin ... **3**
1.4. Second Life ... **4**
1.5. Motivation ... **5**
1.6. Aufgabe ... **5**

2. Entwicklung des weblin Second Life Generator ... **6**
2.1. Entwicklungskonfiguration ... **6**
2.1.1. Hardware ... 6
2.1.2. Software ... 6
2.2. Problemanalyse ... **7**
2.3. Realisationsanalyse und -beschreibung ... **8**
2.4. Realisation ... **11**
2.4.1. Grundlegendes ... 11
2.4.1.1. Programmiersprachen ... 11
2.4.1.2. Datentypen ... 11
2.4.1.3. Dateistrukturen ... 12
2.4.1.4. Speicherverwaltung ... 14
2.4.1.5. Compilereinstellungen ... 14
2.4.1.6. QAvimator ... 15
2.4.1.7. CxImage ... 16
2.4.2. Programmaufbau ... 16
2.4.2.1. Dateien ... 16
2.4.2.2. Programmorganisationsplan ... 18
2.4.2.3. Makros ... 20
2.4.2.4. Hauptprogramm ... 21
2.4.3. Klassen ... 24
2.4.3.1. WGenerator ... 24
2.4.3.2. Threadklassen ... 28
2.4.3.2.1. WHTTPClientWrapper ... 29
2.4.3.2.2. WSavePNGThread ... 29
2.4.3.2.3. WGenXMLThread ... 31
2.4.3.3. Dialogklassen ... 32
2.4.3.3.1. WLoginDialog ... 32
2.4.3.3.2. WProgressDialog ... 32
2.4.3.3.3. WChooseLangDialog ... 33
2.4.3.4. Hilfsklassen ... 33

2.4.3.4.1. BoxDialog 33
2.4.3.4.2. File 33
2.4.3.4.3. glob 34
2.4.3.4.4. Ini 34
2.4.3.4.5. Lang 34
2.4.3.4.6. Tools 35
2.4.3.5. Wichtige verwendete Second Life Klassen 35
2.4.3.5.1. LLAgent 35
2.4.3.5.2. LLBVHLoader 36
2.4.3.5.3. LLKeyframeMotion 36
2.4.3.5.4. LLImageRaw 36
2.4.3.5.5. LLUUID 36
2.4.3.5.6. LLViewerCamera 37
2.4.3.5.7. LLViewerObjectList 37
2.4.4. Wichtige Änderungen an Second Life Dateien 38
2.4.5. Fehlerbehandlung 42

3. Anwendungsfälle 43

4. Zusammenfassung 44

5. Ausblick 45

5.1. weblin Second Life Generator 45
5.1.1. Mögliche Erweiterungen 45
5.1.2. Ansatzpunkte für Optimierungen 45
5.2. Virtuelle Welten allgemein 46
5.3. Virtuelle Präsenzen allgemein 46
5.4. weblin und Second Life 47

6. Benutzerhandbuch 48

6.1. Anforderungen 48
6.2. Installation 48
6.3. Bedienungsanleitung 49
6.3.1. Programmstart 49
6.3.2. Dialoge 50
6.3.2.1. Anmeldedialog 50
6.3.2.2. Fortschrittsdialog 51
6.3.2.3. Sprachauswahldialog 51
6.3.3. Optionen 52
6.3.3.1. Start 52
6.3.3.2. Erstelle Bilder 52
6.3.3.3. Erstelle XML 53
6.3.3.4. Verschicke XML 53
6.4. Fehlermeldungen 54
6.5. Wiederanlaufbedingungen 58

Literaturverzeichnis 59

Schnittstellendokumentation 60

Abbildungsverzeichnis

Abb. 1 Programmablaufplan 19
Abb. 2 Hauptprogramm 21
Abb. 3 Fortschrittsdialogschleife 23
Abb. 4 Programmstart 49
Abb. 5 Anmeldedialog 50
Abb. 6 Fortschrittsdialog 51

Tabellenverzeichnis

Tab. 1 Entwicklungshardware 6
Tab. 2 Entwicklungssoftware 6
Tab. 3 Compilereinstellungen 15
Tab. 4 Systemanforderungen des Programms 48

1. Einleitung

1.1. Virtuelle Welten

Bei virtuellen Welten wird heutzutage an die Fiktionen aus der „Matrix“-Trilogie oder an virtuelle Onlinewelten wie „Second Life“ gedacht. Virtuelle Welten reichen aber noch viel weiter. „Matrix“ beschreibt nur eine idealisierte Form davon. Eine virtuelle Welt in der es den „Bewohnern“ nicht auffällt, dass sie sich in einer solchen befinden.

Virtuelle Welten sind in diesem Sinne alle nicht real existierenden Umgebungen. Diese können computerbasiert sein oder zum Beispiel nur im Kopf eines einzelnen Menschen oder eine Gruppe existieren, wie beispielsweise bei dem Rollenspiel „Das schwarze Auge“.

Computerbasierte, simulierte Welten besitzen für gewöhnlich physikalische oder topologische Eigenschaften der realen Welt und simulieren meist in Echtzeit. Es zählen alle Applikationen zu virtuellen Welten, in denen der Benutzer mit dieser Welt interagieren kann. „Tetris“ oder „Pong“ als virtuelle Welten einzuordnen würde zu weit gehen, aber einfache Spiele wie „Super Mario“ sind durchaus denkbar. Sie stellen eine virtuelle Umgebung dar, in der relativ frei agiert werden kann. Zur Interaktion in der Welt ist es nicht zwingend nötig eine grafische Darstellung in 2D oder 3D zu haben. Virtuelle Welten können auch rein textbasiert sein.

Heute unterscheiden sich die computerbasierten virtuellen Welten grob in drei Kategorien. Einerseits die Singleplayer-Welten, in denen der Benutzer nicht mit virtuellen Charakteren interagieren kann, die von anderen realen Benutzern gesteuert werden. Andererseits zwei Arten der Multiplayer-Welten. Zum einen die Spiele, in denen es klare Ziele zu erreichen gilt. Zum anderen Multiplayer-Welten in denen ohne definiertes Ziel jeder Benutzer selbst entscheidet und wie in der wirklichen Welt frei interagieren kann. Zwischen den beiden Kategorien von Multiplayer-Welten gibt es natürlich jede Art von Abstufung. Der gesamte Bereich der MMOGs[1] ist zwischen diesen beiden Kategorien angeordnet. So ist es in „World of Warcraft“ z.B. unabhängig von den Aufgaben möglich, sich zu bewegen oder

[1] Massive Multiplayer Online Gaming

auch nur mit Mitspielern kommunizieren, sowie sich ein Haus einzurichten, was den Benutzer dem Ziel des Spiels nicht näher bringt.

Die Kommunikation in virtuellen Welten erfolgt in jeder erdenklichen Form: Text, Ton, Gestik oder Mimik des Repräsentanten des Benutzers in der Welt.

1.2. Virtuelle Präsenzen

Virtuelle Präsenzen sind hauptsächlich Präsenzen in der virtuellen Welt des World Wide Web. Dabei werden Personen die sich zur gleichen Zeit auf der gleichen Webseite bewegen für den jeweils Anderen sichtbar gemacht. Dies kann rein textbasiert geschehen oder aber durch Figuren, die den jeweiligen Benutzer repräsentieren. Im Vordergrund steht hier die Kommunikation der Benutzer, die sich auf dieser Seite aufhalten. Diese Kommunikation kann per Text, Video oder Audio sein.

Ein weiterer wesentlicher Aspekt der virtuellen Präsenz ist, dass diese dem wirklichen Leben möglichst nahe nachempfunden werden soll. Es sollen nur an den Orten, hier Webseiten, andere Personen zu sehen sein, die sich zur gleichen Zeit auf derselben Webseite befinden. Als Beispiele:

- Eine Person betritt eine Webseite eines Versandhauses. Hier befindet sich eine für sie sichtbare andere Person, die einen Servicemitarbeiter der Webseite darstellt. Die Beratung des Kunden kann so direkt erfolgen, wie in einer Filiale.
- Der Autor einer privaten Webseite trifft auf die Besucher dieser Seite, die wahrscheinlich seine Freunde sind oder sich für auf der Webseite dargestellte Themen interessieren. Analog dazu können Freunde nachhause eingeladen werden oder bei der Gartenarbeit eine Unterhaltung mit Passanten über Floristik geführt werden.

Diese Beispiele lassen sich beliebig fortführen und sind in sehr weitem Sinne aus dem wirklichen Leben in die virtuellen Präsenzen und virtuellen Plätze übertragbar.

1.3. weblin

Weblin ist ein Chat-Client und ein Internetportal, das versucht sich sehr stark an dem Idealtyp von virtuellen Präsenzen zu orientieren. Der Client ermöglicht es andere Benutzer, die sich zur gleichen Zeit auf der gleichen Webseite aufhalten zu sehen und mit Ihnen kommunizieren. Für die Kommunikation wird das offene Jabber[2]-Protokoll verwendet. Alle nötigen Daten um die jeweils andere Präsenz in Form von Text und Bild darzustellen werden über dieses Formular übertragen. Der Text für die Kommunikation, ID[3], Benutzername und ähnliches wird direkt übertragen. Um Bilder darzustellen bekommen die anderen Personen auf derselben Seite nur die URIs[4] um sich die Bilddaten eigenständig vom weblin Server zu laden. Die Figuren werden bisher als Standbilder oder animierte GIFs[5] dargestellt und können beliebiger Art sein, wie z.B. Tiere, Menschen, Fantasiewesen oder ein Foto des Benutzers. Da weblin browserbasiert ist, haben die Figuren maximal zwei Aktionsdimensionen, von denen eine noch nicht verfügbar ist, da die Avatare sich immer auf der Statusleiste bewegen müssen.

[2] Link zur offiziellen Webseite: http://www.jabber.org/

[3] Eindeutige Identitätsbezeichnung, meist numerisch

[4] Unique Resource Identifier, Link: http://www.w3.org/MarkUp/html3/terms.html

[5] Graphics Interchange Format

1.4. Second Life

Second Life ist heute eine der bekanntesten virtuellen Online-Welten in der freies nicht zwangsweise zielgerichtetes Interagieren möglich ist. Sie wird von dem Unternehmen Linden Labs betrieben. Die Figuren werden dreidimensional dargestellt und können sich frei in der Welt und im dreidimensionalen Raum bewegen. Sie besitzen somit zwei Aktionsdimensionen mehr als in weblin. Dabei können Sie ebenso wie in weblin beliebiger Art sein, sind aber bis auf einen verschwindend geringen Anteil humanoid. Die Benutzer können per Chat oder Voice-Chat kommunizieren. Jeder Benutzer von Second Life kann Objekte erstellen, damit handeln, Land mieten oder kaufen und vieles mehr. Die Welt selbst ist in Regionen aufgeteilt, in denen die komplette Umgebung durch den Besitzer bestimmt wird. Ebenso ist es in Second Life möglich beliebige Objekte zu erstellen und diesen z.B. mit Hilfe von Skripten bestimmte Funktionalitäten aufzuprägen.

1.5. Motivation

Es gibt enorm viele, teilweise sehr unterschiedliche, virtuelle Welten. Diese werden in absehbarer Zukunft an ihre Wachstumsgrenzen stoßen, da sie nicht alle Möglichkeiten von virtuellen Welten in einer abbilden und somit nicht mehr Benutzer binden können. Die Benutzer wollen sich aber in Zukunft frei zwischen den verschiedensten Arten virtueller Welten bewegen. Ihre Figur, im folgenden als Avatar bezeichnet, die Ausrüstung und das Geld aus den Welten wollen sie dabei natürlich mitnehmen und überall verwenden können. Diese angenommenen Verhaltensweisen sind vergleichbar mit jemandem der im wirklichen Leben aus seinem Heimatland auswandert. Auf diesen Trend hin haben sich Linden Labs[6] und das IT-Unternehmen IBM[7] entschlossen gemeinsam Standards[8] zu entwickeln, die das Wechseln zwischen virtuellen Welten, die sich an diese Standards halten, ermöglichen sollen.

Virtuelle Präsenz ist vergleichbar mit der virtuellen Welt des Internets, in der sich wesentlich mehr Personen bewegen als in jeder anderen virtuellen Welt. Auch in diesen speziellen virtuellen Welten besteht der Wunsch der Benutzer, ihren Avatar auch in diese virtuelle Welt mitnehmen und dort mit Anderen kommunizieren zu können. Das bedeutet, dass hier idealer Weise auch Standards benötigt werden. Diese sind aber wesentlich schwerer umzusetzen, da diese virtuellen Welten in ihrer Beschaffenheit enorm unterschiedlich sind.

1.6. Aufgabe

Um aber zu überprüfen, ob Benutzer von barrierefreiem Verwenden von Avataren überhaupt Gebrauch machen würden und ob etwas derartiges auch dauerhaft angenommen wird, muss dies an einem Beispiel überprüft werden. An dieser Stelle setzt diese im Rahmen eines Praktikums bei der Zweitgeist GmbH entstandene Bachelor-Thesis an. Es soll eine Konvertierung zwischen den sehr populären Programmen für virtuelle Welten und virtuelle Präsenzen, Second Life und weblin, stattfinden, so dass die Benutzer ihre virtuellen Repräsentanten aus Second Life weiter verwenden können.

---

[6] Link zur Unternehmensseite: http://lindenlab.com/

[7] International Business Machines, Link zur Unternehmensseite: http://www.ibm.com/

[8] Link: http://www.ibm.com/press/us/en/pressrelease/22428.wss

2. Entwicklung des weblin Second Life Generator

2.1. Entwicklungskonfiguration

2.1.1. Hardware

Komponente	Eigenschaften
Prozessor	Intel Core2 Duo CPU E6550 2.33 GHz
Arbeitsspeicher	1.0 GB DDR2 333 MHz RAM
Mainboard	Asus P5K SE mit Intel Bearlake P35 Chipsatz
Grafikkarte	ATI Radeon X1550, 512 MB
Netzwerk	10/100/1000 Ethernet

Tab. 1 Entwicklungshardware

2.1.2. Software

Windows XP Professional Version 2002 Service Pack 2
Microsoft Visual Studio .NET Professional 2003
Microsoft .NET Framework 1.1 Service Pack 1
Microsoft Plattform SDK 2003
Microsoft DirectX SDK (Dezember 2006)
Nullsoft Scriptable Install System
ActivePython 2.5.1.1
QAvimator[9]

Tab. 2 Entwicklungssoftware

[9] Vgl. Kapitel 2.4.1.6

2.2. Problemanalyse

Bei dieser recht groben aber doch sehr speziellen Aufgabenstellung treten diverse Problemfälle auf, die es zu lösen gilt. Figuren in Second Life oder weblin, im Folgenden Avatare genannt, bestehen bei Second Life aus einer Definition für das Aussehen des Avatar selbst, welches im so genannten Genpool beschrieben wird und aus seinen Attachments. Dies sind die in Second Life verwendeten Objekte, die ein Avatar „trägt“, beispielsweise Gegenstände wie Kleidung, Schmuck, eine Perücke oder Vergleichbares. Diese Attachments können zusätzlich noch Eigenschaften haben, die durch Skripte definiert sind, die über die LSL, die „Linden Scripting Language“, beschrieben werden können.

Das erste und vorrangige Problem besteht in der Datenbeschaffung, da diese nicht einzeln, direkt und beliebig vom Second Life Server abgefragt werden können. Die Daten, die zur Darstellung eines Avatar benötigt werden, umfassen eine enorme Größe. Dabei gilt es das Problem zu lösen, wie der Benutzer gleichzeitig das Programm verwenden und sich ohne Einschränkungen im Internet aufhalten kann.

Vorausgesetzt die Daten sind vollständig bei allen Benutzern des weblin Clients angekommen, stellt sich die Frage, wie diese visualisiert werden sollten. Die Bewegungen sind kinetische Änderungen der einzelnen Gelenke und es ist möglich an diesen Gelenken oder den einzelnen Körperteilen Attachments zu platzieren. Da es nicht sehr komfortabel ist sich im Internet fortzubewegen, wenn permanent eine Applikation viele Ressourcen verwendet, sollte die Applikation möglichst minimale Ressourcen im Sinne von Arbeitsspeicher und Rechenleistung benötigen. Zu diesen zu lösenden Problemen kommt hinzu, dass gewährleistet werden muss, dass Benutzer nicht den Avatar eines Anderen verwenden können. Zudem das Aktionsdimensionsproblem, da Avatare in Second Life in drei Dimensionen interagieren können, in weblin hingegen nur in einer Dimension.

Die Zielgruppe für das Programm ist sehr weit gefächert, da weblin wie auch Second Life eine sehr breite Zielgruppe ansprechen und sich diese zu einem Großteil überschneidet. Diese Schnittmenge ist die Zielgruppe des zu entwickelnden Programms, also in diesem Sinne jede Person, die das Internet nutzt und über einen ausreichend schnelle Verbindung verfügt.

2.3. Realisationsanalyse und -beschreibung

Ideal wäre es einen kompletten eigenen Client für Second Life zu programmieren, der als Avatar-Plugin für den weblin fungiert. Dieser Client bezieht im besten Fall alle notwendigen Daten um einen beliebigen Avatar darzustellen direkt vom Second Life Server. Dafür ist eine Beschreibungsdatei der Charaktereigenschaften nötig, die Daten von Größe über Hautfarbe bis hin zur Ausrichtung der Augenbrauen enthält. Außerdem ist eine Liste aller Attachments erforderlich, die dieser Avatar trägt mit der Zusatzinformation an welcher Position diese getragen werden. Zusätzlich ist es notwendig die Texturen für den Avatar sowie die seiner getragenen Attachments vom Server zu beziehen.

Über den weblin Server würde anstatt der VPI[10] Daten über die Position der Avatare in der weblin Datenbank, wie es bei „normalen“ eigenen Avataren der Fall ist, nur der Name der Second Life Figur über die Konfigurationsdatei kommuniziert werden. Das Plugin würde je weblin Client den vollständigen Bezug der Daten vom Second Life Server, sowie deren Verarbeitung und Visualisierung übernehmen. Diese Avatare würden als dreidimensionale Modelle innerhalb des weblin Clients verarbeitet und dargestellt werden.

Die Umsetzung auf diese Weise ist in dem beschriebenen Umfang nicht möglich, da die Kenntnis über die Struktur der Daten, sowie der Kommunikation nicht gegeben und auch in dem gegebenen Zeitrahmen nicht zu erlangen und umzusetzen ist. Die Daten können nicht direkt vom Second Life Server abgefragt werden. Es ist nur möglich die Daten der Avatare und Objekte einer Region zu erhalten, wobei die Reihenfolge serverseitig bestimmt wird und es somit ausgeschlossen ist alle Benutzer des weblin Client in eine Region der Second Life Welt zu teleportieren. Zumal die Second Life Server bei über einhundert Benutzern eine Leistungsgrenze erreichen.

Eine weitere Möglichkeit besteht darin, die eigenen Daten vom Second Life Grid[11] zu beziehen und diese direkt an alle weblin Benutzer die dieses Plugin besitzen und sich in demselben Chatraum befinden, zu schicken. Da dies aber bedeuten würde, dass je nach Avatar und Anzahl der Benutzer in dem Raum mehrere Megabyte an

[10] Virtual Path Identifier

[11] Hostservergruppe einer Second Life Welt, Link: https://wiki.secondlife.com/wiki/Glossary

einen Neuankömmling geschickt würden, aber vor allem dieser Benutzer seine Daten an eine beliebige Zahl andere Benutzer hätte schicken müssen, ist dies ebenfalls nicht praktikabel. Die Verbindungen würden dabei zu stark ausgelastet.

Zudem wären selbst solche Plugins zu groß, um sie mit dem weblin Client als solche auszuliefern, da Basisdaten des Second Life Clients nicht vom Server bezogen werden sondern mit dem Client ausgeliefert werden. Das sind unter anderem mehrere Megabyte an Texturdaten.

Aufgrund der dargestellten Restriktionen soll die Funktionalität einen Avatar von Second Life nach weblin zu portieren durch eine eigenständige Applikation gelöst werden. Diese Applikation ist auf Grund der wenig dokumentierten Kommunikation von Second Life Client und Server, sowie der internen Datenstrukturen eher auf Basis des Second Life Clients selbst zu erstellen. Weblin besitzt bereits ein Plugin, welches animierte Avatare im GIF-Format darstellt. Es ist möglich, auf der Webseite mit eigenen Animationen eigene Avatare zu erstellen, die serverseitig gespeichert werden und direkt vom weblin Server abgerufen werden. Dementsprechend können die Avatare auf Basis des Second Life Clients herunter geladen, animiert und die einzelnen Animationen als animierte Bilder auf den weblin Server hochgeladen werden, um diese dort wie einen selbst erstellten Avatar zu behandeln. Durch die GIF-Formatierung und das einmalige Hochladen vom Benutzer werden die Datenmengen, die kommuniziert werden, immens verringert. Die Visualisierung soll dabei einmalig in 3D durch den Second Life Client erfolgen. In weblin selbst werden nur die animierten Grafiken ebenso behandelt wie hauseigene und durch das GIFAvatar-Plugin visualisiert. Die Kommunikation der Bilder findet dabei von weblin über bestehende Formate statt.

Bliebe noch das Problem der Animation zu lösen. Es gibt vordefinierte Animationen in Second Life. Diese werden serverseitig kommuniziert, es ist aber möglich eigene Animationen hoch zu laden. Da die Avatare in weblin relativ klein sind, einen maximalen Aktionsbereich von 100 mal 100 Pixel haben und sich nur im zweidimensionalen Raum bewegen können, müssen eigene Animationen erstellt werden, die sich auf einen solchen Aktionsbereich beschränken, um die Größe der Darstellung des Avatars zu erhöhen. Diese Animationen müssen selbstständig im modifizierten Second Life Client abgespielt werden, da sie nicht für andere Benutzer von Second Life sichtbar sein und zudem nichts kosten sollen. Das Hochladen von Animationen auf den Second Life Server ist aber kostenpflichtig.

Es stellt kein Problem dar auf Basis des Second Life Clients zu arbeiten, da dieser unter der GPL[12] steht und somit „open source“ ist. Es ist in diesem Fall auch nicht mehr möglich sich als fremde Identität auszugeben, da zum Generieren der Bilder die Authentifizierung am Second Life Server notwendig ist.

Das Problem der Aktionsdimensionen wir in jedem Fall in der Art gelöst, dass die Animation aus der gewählten Kameraperspektive nur zweidimensional erfolgt. Die verwendeten Animationen müssen dementsprechend darauf ausgelegt sein den Avatar nicht in die Tiefe zu bewegen. In der Höhe darf der Avatar nur im Rahmen der 100 mal 100 Pixel bewegt werden.

[12] General Public Licence

2.4. Realisation

2.4.1. Grundlegendes

2.4.1.1. Programmiersprachen

Aus der Realisationsanalyse geht hervor, dass die Aufgabe durch eine eigenständige Applikation zu lösen ist.

Unabhängig von der ausgewählten Realisation wird die Programmiersprache C++ verwendet, da sowohl der Second Life Client als auch der weblin Client in C++ geschrieben sind und bestehende Klassen somit problemlos wieder verwendet werden können. Für die serverseitige Verarbeitung wird PHP, der „PHP Hypertext Preprocessor“, verwendet und das bestehende Portal um dieses Skript erweitert. Da für die eigenständige Applikation auf Basis des Second Life Clients gearbeitet wird, ist für die Generierung der Installationsdatei die Skriptsprache Python vorgegeben.

2.4.1.2. Datentypen

Mit einer Ausnahme gibt es keine eigenen Datentypen, da die Programmstruktur ansonsten komplett objektorientiert aufgebaut ist. Diese eine Ausnahme ist der *TAnimation* Datentyp. Er repräsentiert eine Animation und enthält die Teilbilder als Zeiger auf eine Standardliste mit LLImageRaw[13]-Objekten, den Animationsschlüssel als LLUUID[14] um die Animation aufzurufen, einen vorzeichenlosen 32 Bit Wert mit der Anzahl der Bilder, die für diese Animation gerendert werden müssen und den Namen der Animation als nullterminierte Zeichenkette.

```
struct TAnimation {
  std::list<LLImageRaw*>* pImageList;
  LLUUID animID;
  U32 nAnimSteps;
  char* szAnimName;
};
```

[13] Vgl. Kapitel 2.4.3.5.4
[14] Vgl. Kapitel 2.4.3.5.5

2.4.1.3. Dateistrukturen

Alle Dateien, die hier behandelt werden, werden während des Programmablaufs erstellt oder geladen. Zudem sind sie ausschließlich für die dem Second Life Client hinzugefügten Funktionalitäten und wurden nicht von Linden Labs definiert.

Die Dateiformate „.ini“ und „.lang“ sind gleich aufgebaut. Sie bestehen aus Schlüssel-Wert-Paaren, die durch Zeilenumbrüche getrennt sind. Zwischen Schlüssel und Wert befindet sich immer ein Gleichheitszeichen. Im „.lang“ Format werden die Schlüssel generell nur in Großbuchstaben, getrennt von Unterstrichen geschrieben. Zusätzlich gibt es im „.lang“ Format die Möglichkeit im Wert eines Paares Wörter zu definieren, die zu ersetzen sind. Dazu wird der zu ersetzende Wert wie ein Schlüssel in Großbuchstaben geschrieben und von doppelten eckigen Klammern umrahmt.
Beispiel:

```
DIALOG_CAPTION=[[PROG_NAME]] Figurgenerator
```

Das XML[15] Dateiformat für Gestiken („gestures.xml“, „gesturesFemale.xml“) besitzt wie die erweiterte Avatarkonfigurationdatei („send.xml“) eine sehr flache XML Struktur. Das Format für Gestiken besitzt genau ein *gestures_def* Tag in dem beliebig viele *gesture* Tags vorkommen können. Diese *gesture* Tags besitzen die folgenden Attribute

- *name*: der Name der Animation, der in jeder Datei eindeutig sein muss
- *capture_steps*: die Anzahl der Renderingschritte bis diese Animation abgebrochen werden soll, wobei null bedeutet, dass die Animation bis zu ihrem Ende abgespielt wird

[15] eXtensible Markup Language, Link: http://www.w3.org/XML/

Zusätzlich zu den beiden genannten Attributen muss eines der beiden Attribute *anim_file* oder *anim_name* im Tag enthalten sein. *anim_file* sorgt dafür, dass eine erstellte Animationsdatei („.bvh“[16]) geladen wird, wohingegen *anim_name* eine vordefinierte Animation von Second Life ausführt.
Beispiel:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<gestures_def>
 <gesture name="idle" anim_file="stand_1.bvh" capture_steps="1" />
 <gesture name="idle1" anim_name="stand_1" capture_steps="0" />
```

Die erweiterte Avatarkonfigurationsdatei wird beim Generieren der XML Datei für das Hochladen zum weblin Server erzeugt. Sie enthält einen umschließenden *config* Tag mit einem *version* Attribut, welches die Version der Konfigurationsdatei angibt. Dieser Tag enthält beliebig viele *param* und *sequence* Tags. Die *param* Tags besitzen ein *name* und ein *value* Attribut stellen also einfache Schlüssel-Wert-Paare dar. *sequence* Tags sind umfangreicher, sie besitzen die Attribute *group*, *name*, *probability*, *type*, *in* und *out* und hier als Inhalt zusätzlich noch *image* Tags, die ein Attribut *num* besitzen, welches die Nummer des Bildes in der jeweiligen Sequenz beschreibt. In dem *image* Tag befinden sich die Daten des Bildes Base64 enkodiert. Die Attribute *group* und *name* des *sequence* Tags sind gleichzusetzen; sie enthalten den Animationsnamen. *probability* ist die Wahrscheinlichkeit mit der diese Animation abgespielt wird, wenn es mehrere Animationen für dieselbe Aktion gibt, wie bei *idle* oder *chat*. Das *type* Attribut gibt an, ob die Animation ein Status, eine Basisanimation, eine Geste oder eine Übergangsanimation zwischen zwei Status ist, z.B. vom Stehen zum Sitzen, definiert. Basis- und Statusanimationen werden beim Betreten des Raumes geladen, Gesten und Übergangsanimationen auf Anfrage, wenn diese ausführt werden. *in* und *out* definieren, welcher Status vor bzw. nach der Animation gilt. Die normale Avatarkonfigurationsdatei besitzt anstatt mehrerer *image* Tags nur ein *animation* Tag, welches ein *src* Attribut besitzt, in dem sich der absolute Pfad zu dem Bild befindet.

[16] Vgl. Kapitel 2.4.1.6

2.4.1.4. Speicherverwaltung

Es wird im Besonderen darauf geachtet, dass kritischer Speicher oder große Mengen an Speicher sauber wieder freigegeben werden. So müssen z.B. alle GDI[17]-Objekte wieder freigegeben werden, da die Applikation ansonsten nach wenigen Sekunden so viele dieser Objekte besitzen kann, dass keine andere Anwendung mehr arbeiten kann. Außerdem müssen alle generierten Bilder sowie deren Kopien komplett freigegeben werden, da die Auslagerungsdatei ansonsten sehr schnell vollständig belegt ist.

Die Speicherverwaltung wird ansonsten durch die Objektorientierung erreicht. Es gibt z.B. die „glob“ Klasse, die ein Singleton[18] ist und somit nur eine Instanz dieser Klasse während der gesamten Programmlaufzeit existiert.

Auf die „Smartheap“-Bibliothek, die von Second Life verwendet wird, wurde bewusst verzichtet, da diese proprietär ist. Die Speicherverwaltung wäre mit dieser Bibliothek nur minimal besser.

2.4.1.5. Compilereinstellungen

Das Microsoft Visual Studio 2003 Projekt „newview“[19] ist in der von Linden Labs mitgelieferten Projektmappe als Startprojekt definiert und gibt somit alle Compileroptionen für die anderen Projekte vor. Hier werden nur die wichtigsten Einstellungen behandelt. Alles Weitere kann in den beiliegenden Projektdateien eingesehen werden.

[17] Graphics Device Interface
[18] Link: http://www.fhwedel.de/~si/vorlesungen/softwaredesign/Erzeugungsmuster/Singleton.html
[19] Vgl. Kapitel 2.4.2.1

Allgemein	
Konfigurationstyp	Anwendung
MFC und ATL	Nicht verwenden, da nicht benötigt
Zeichensatz	Unicode (ucs-2)
Compiler	
Warnungsebene	Level 3 (/W3)
Warnungen als Fehler behandeln	Ja (/WX)
Optimierung	Für Geschwindigkeit (/O2), ansonsten für nichts optimieren.
Laufzeitbibliothek	Multithreaded (/MT)
Vorkompilierte Header erstellen/verwenden	Ja (/Yu), wird für Second Life benötigt
Linker	
Ausgabedatei	wSLGen.exe
Subsystem	Nicht festgelegt, für Intel/AMD Support
Zielcomputer	Nicht festgelegt, für Intel/AMD Support

Tab. 3 Compilereinstellungen

Aus den Projektdateien werden die Projekte „llaudio“ und „llmedia“ entfernt, sowie die Bibliotheken FMOD[20], KDU[21], Smartheap und Quicktime, da diese nicht benötigt werden. Außerdem sind sie proprietär sind und stehen somit nicht unter der GPL. Des Weiteren dürfen sie nicht ohne die zugehörigen Lizenzen weitergegeben werden.

2.4.1.6. QAvimator

QAvimator[22] ist eine Applikation, die es ermöglicht BVH[23] Animationen zu erstellen und zu bearbeiten. Sie ist zwar noch im Alpha-Status, aber ausreichend mächtig. Der Vorteil dieser Applikation ist, dass sie speziell für Second Life Animationen ausgelegt ist. Aufgrund dessen bietet sie zwei Modelle, für welche die jeweiligen Keyframe-Eigenschaften gesetzt werden können. Komplette erstellte Animationen lassen sich so als BVH-Dateien abspeichern und in Second Life verwenden.

[20] Audio Bibliothek, Link zur Herstellerseite: http://www.fmod.org/

[21] JPEG2000 Bibliothek, Link zur Herstellerseite: http://www.kakadusoftware.com/

[22] Link zur Herstellerseite: http://www.qavimator.org/

[23] Bounding Volume Hierarchies

2.4.1.7. CxImage

CxImage ist eine offene C++-Bildbibliothek zum Laden, Speichern, Anzeigen und Transformieren von Bilddaten z.B. in den Formaten PNG[24] oder GIF. Sie bietet somit die Möglichkeit Rohdaten zum Beispiel als PNG abzuspeichern und wird daher verwendet, um die erstellten Bilder für den Transport zu optimieren.

2.4.2. Programmaufbau

2.4.2.1. Dateien

Bei der Ordnerstruktur wird die durch Second Life vorgegebene verwendet. So sind „libraries“ und „indra“ als wesentliche Ordner zu nennen. „libraries“ enthält alle externen Bibliotheken und ihre Header-Dateien, die von Second Life verwendet werden. „indra“ ist der Ordner, welcher alle Quellcode-Dateien von Second Life selbst enthält und in Unterordner nach Funktionalität strukturiert ist. Je Unterordner existiert ein eigenständig kompilierbares Projekt. Zusätzlich gibt es einen Unterordner, der eine Visual Studio Projektmappendefinition enthält, in der alle Einzelprojekte enthalten sind. Kurzbeschreibungen zu den einzelnen Projekten sind im Second Life Wiki[25] zu finden. Innerhalb der Objektordner existiert für die Quelldateien keine weitere Struktur. Es werden lediglich bei Bedarf Ausgabeordner erstellt.

Von den existierenden Projekten werden „llaudio“ und „llmedia“ entfernt, da diese nicht benötigt werden.

Das Hauptprojekt, welches auch den Programmeinstiegspunkt darstellt, nennt sich „newview“ und besitzt eine erweiterte Unterstruktur, die neben den Ausgabeordnern noch zusätzliche enthält, um die Ausführung des Programms für Debuggingzwecke zu ermöglichen, sowie zusätzliche Dateien und Ordner um diese dem Installationsprogramm hinzuzufügen. Dieses Projekt enthält ebenfalls alle selbst erstellten Dateien. Dabei besitzen die selbst erstellten Dateien, im Gegensatz zu denen von Second Life, nicht das Präfix „LL“ für Linden Labs, sondern „W“ für weblin. Diese Dateien, die aus dem weblin Client übernommen wurden, bilden die Basisfunktionalitäten ab.

---

[24] Portable Network Graphic

[25] Vgl. Viewer Source, Link: http://wiki.secondlife.com/wiki/Viewer_source

Aus dem weblin Client:

- wHTTPClient: einfacher HTTP[26] Client für einmalige Anfrage.
- wSElem: Strukturen und Klassen für Schlüssel-Wert-Paare als Liste.
- wSString: verwaltet UTF-8[27] Strings.
- wSSystem: Speicher und Zeigerverwaltung.
- wStdAfx: lädt einige benötigte Windows Header Dateien.
- wThread: abstrakte Darstellung eines Windows Threads.

Im Rahmen dieser Arbeit erstellte Dateien:

- wAvatarGenerator: Hauptprogramm.
- wChooseLangDialog: Sprachauswahldialog.
- wDialog: Basis Windowsdialog.
- wGenerator: Controller der Programmfunktionalität.
- wGenXMLThread: lädt Bilder und generiert eine XML-Datei.
- wHTTPClientWrapper: kapselt eine HTTP-Anfrage.
- wLoginDialog: zeigt den Anmeldedialog an.
- wProgressDialog: zeigt den Fortschrittsdialog an.
- wSavePNGThread: optimiert und speichert Bilder.
- wTools: kleine Hilfsklassen.
- wTypes: Makros, Typen und Aufzählungen.
- wPreprocessor: Makros für die vorkompilierten Header.

[26] HyperText Transfer Protocol
[27] 8-Bit Unicode Transformation Format

2.4.2.2. Programmorganisationsplan

Der Programmorganisationsplan ist in Abb. 1 auf der folgenden Seite dargestellt. Folgende Anmerkungen dienen dem besseren Verständnis und der Nachvollziehbarkeit der Darstelung:

Die „wTools“ Datei ist eine Sammlung von Hilfsklassen, die von allen anderen Programmteilen verwendet wird. In ihr ist die „glob“ Klasse das zentrale Element. Im Programmorganisationsplan wurde bewusst auf eine genauere Darstellung der Zugriffe von außerhalb der „wTools“ verzichtet um die Übersicht zu gewahren.

Die Threads, die von der abstrakten „Thread“ Klasse erben werden von „WGenerator“ aufgerufen. Die Pfeile sind hier beidseitig, da die Threadklassen über Callbackmethoden den WGenerator aufrufen.

Ebenso wie bei der abstrakten „Thread“ Klasse bedeutet es bei der abstrakten „WDialog“ Klasse, dass die erbenden Klassen Zugriff auf die „wTools“ haben.

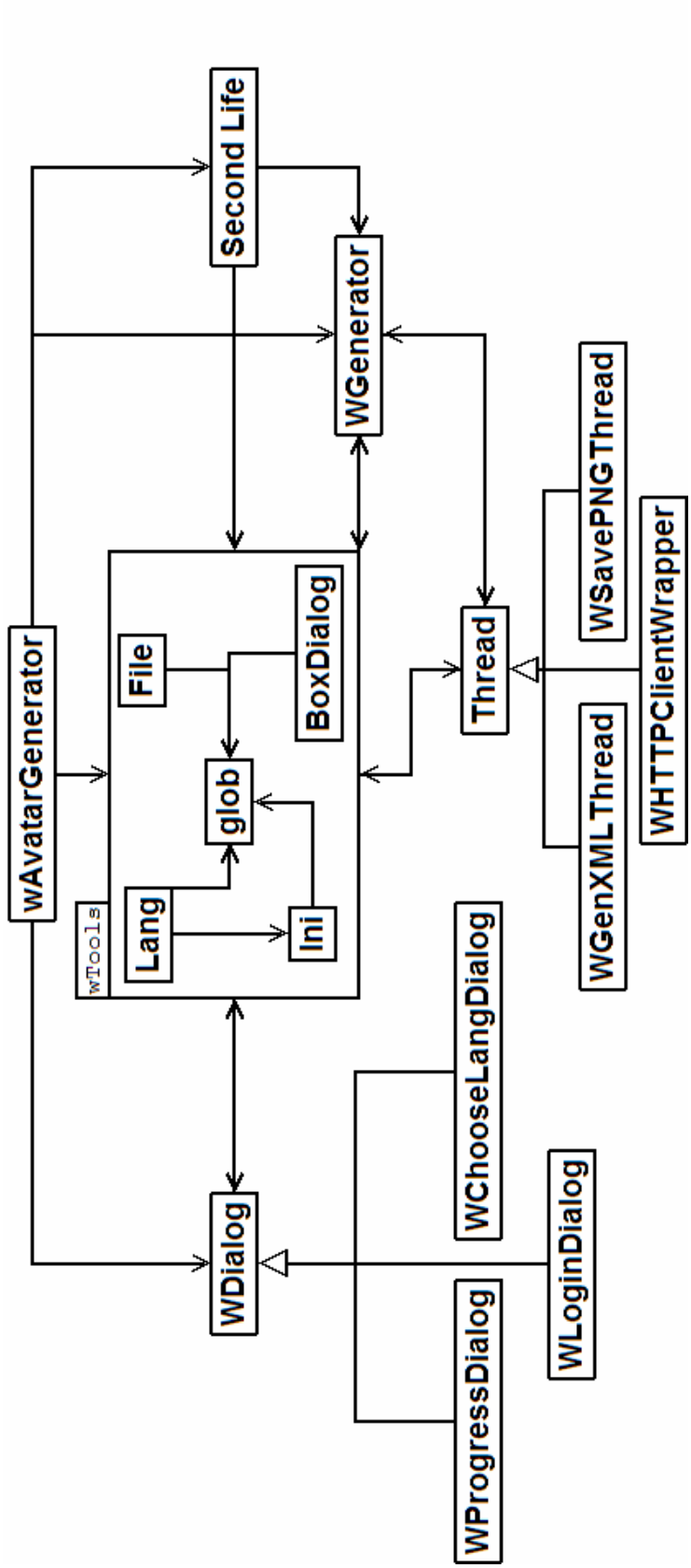

Abb. 1 Programmablaufplan

2.4.2.3. Makros

In den folgenden Abschnitten werden die verwendeten Makros nach ihrer Funktion geordnet dargestellt. Die Compiler-Flag Makros verhindern, dass Quellcode für Audio und Video mitkompiliert, sowie dass die Umgebung in Second Life angepasst wird.

```
WEBLIN_AUDIO
WEBLIN_VIDEO
WEBLIN_ENV
```

Die „PREVIEW_WINDOW“ Makros beschreiben Position und Ausdehnung des OpenGL[28] Fensters im Fortschrittsdialog, welches die Voransicht für die gerenderten Bilder des Avatar anzeigt.

```
PREV_WINDOW_WIDTH
PREV_WINDOW_HEIGHT
PREV_WINDOW_X
PREV_WINDOW_Y
```

Analog zu den „PREVIEW_WINDOW“ Makros sind die „SNAP_WINDOW“ Makros für das Rendern in einer höheren Auflösung zuständig. In dem durch diese Makros definierten Fenster werden die hochaufgelösten Bilddaten gerendert.

```
SNAP_WINDOW_WIDTH
SNAP_WINDOW_HEIGHT
SNAP_WINDOW_X
SNAP_WINDOW_Y
```

Anzahl der Bildkanäle zum Rendern. Für RGBA dem entsprechend vier Kanäle.

```
SNAP_CHANNELS
```

Diese Makros beschreiben Farbwerte für die Progressbar wenn die Standardfarbe, die blau ist, abgeändert wird. „OK“ enthält den Farbcode für grün und „NOTOK“ den für rot.

```
COLOR_OK
COLOR_NOTOK
```

[28] Open Graphics Library, Link: http://www.opengl.org/

2.4.2.4. Hauptprogramm

Das Hauptprogramm bezeichnet hier den Programmeinstiegspunkt, sowie von diesem aufgerufene Methoden, die nicht Bestandteil von Klassen sind. Bei Programmstart werden zuerst die Windows „Common Controls“ initialisiert um die Fortschrittsbalken und Listboxen verwenden zu können. Die „RichEdit“ DLL[29] wird hinzu geladen. Wenn das Initialisieren und Laden erfolgreich verlaufen ist wird damit fortgefahren die eigene Initialisierungsdatei[30] zu laden um die eingetragene Sprache[31] zu ermitteln, um diese zu laden. Nach diesen initialen Schritten können der Fortschrittsdialog[32] und der Anmeldedialog[33] erstellt und initialisiert werden.

Wenn bis zu diesem Punkt keine Fehler aufgetreten sind, wird die Hauptschleife betreten, die die beiden Dialoge abwechselnd anzeigt. Dies geschieht solange, wie der Benutzer das Programm nicht beendet oder dieses durch Auftreten eines Fehlers[34] beendet wird.

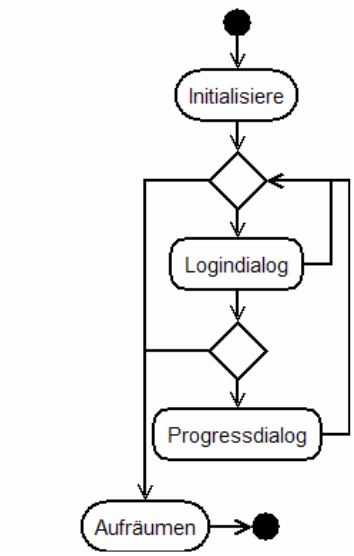

Abb. 2 Hauptprogramm

Beide Dialoge werden in einzelnen Funktionen speziell behandelt und besitzen ein eigene Schleife, die in beiden Fällen die Systemmitteilungen verarbeitet und über

[29] Dynamic Link Library, http://www.microsoft.com/technet/archive/wfw/7_agloss.mspx
[30] Vgl. Kapitel 2.4.3.4.4
[31] Vgl. Kapitel 2.4.3.4.5
[32] Vgl. Kapitel 2.4.3.3.2
[33] Vgl. Kapitel 2.4.3.3.1
[34] Vgl. Kapitel 6.4

die Callback-Funktionen an die Klassen weiterreicht. Als Beispiel wären hier das Zeichnen des Dialogs als Ereignis oder die Tastatur- oder Mauseingabe bei aktivem Dialogfenster zu nennen.

In der Verarbeitungsschleife des Fortschrittdialogs findet zusätzlich zu der Verarbeitung der Mitteilungen, die vom System kommen, noch eine Vorverarbeitung der Bearbeitung der eigentlichen Funktionalität statt, welche ausgeführt werden soll. Im speziellen Falle des Durchlaufens der kompletten Programmfunktionalität durch Drücken des Startknopfes[35] auf dem Anmeldedialog werden bei jedem Schleifendurchlauf, der durch Rendervorgänge sehr unterschiedlich lange dauern kann, zuerst die Dialogereignisse, wie Starten der Animation, verarbeitet. Darauf wird direkt eine Dialogfunktion aufgerufen, die feststellt, ob ein Vorschaubild bereits existiert. Wenn dieses existiert, wird es auf dem Dialog angezeigt. Je nach Auswahl wird nun direkt das Generieren der XML Datei oder das Hochladen dieser Datei über die „run“-Methode des WGenerator[36] gestartet, wenn im Anmeldedialog „Erstelle XML“ oder „Verschicke XML“ ausgewählt wurde. Anderenfalls wird nun ein Timeout gesetzt um die Anzeige des Windows Dialogs zu initialisieren, da darauf die ehemalige Programmeinstiegsmethode des Second Life Clients aufgerufen wird. Es ist nicht möglich diese in einem eigenen Thread auszuführen, daher wird die Anwendung blockiert solange die „SLMainStart“[37] Methode ausgeführt wird. Sobald das Laden des Second Life Client abgeschlossen ist, wird bei jedem Schleifendurchlauf die „SLMainLoop“ Methode aufgerufen. Diese ist eine vereinfachte Variante der Hauptschleife des Second Life Clients. Sie enthält aber im Gegensatz zu dieser keine eigene Schleife, da diese schon durch die umfassende Methode gegeben ist. Wenn ein Fehler aufgetreten ist oder der Prozess erfolgreich durchlaufen wurde, wird die Fortschrittsdialogschleife verlassen und der Anmeldedialog angezeigt, sofern das Programm nicht auf Grund eines Fehlers beendet werden muss. In jedem Fall werden beim Aufräumen die erstellten Objekte und der alloziierte Speicher, als auch die von Second Life verwendeten Ressourcen wieder über die „SLMainEnd“ Methode freigegeben.

[35] Vgl. Kapitel 6.3.3.1
[36] Vgl. Kapitel 2.4.3.1
[37] Vgl. Kapitel 2.4.4

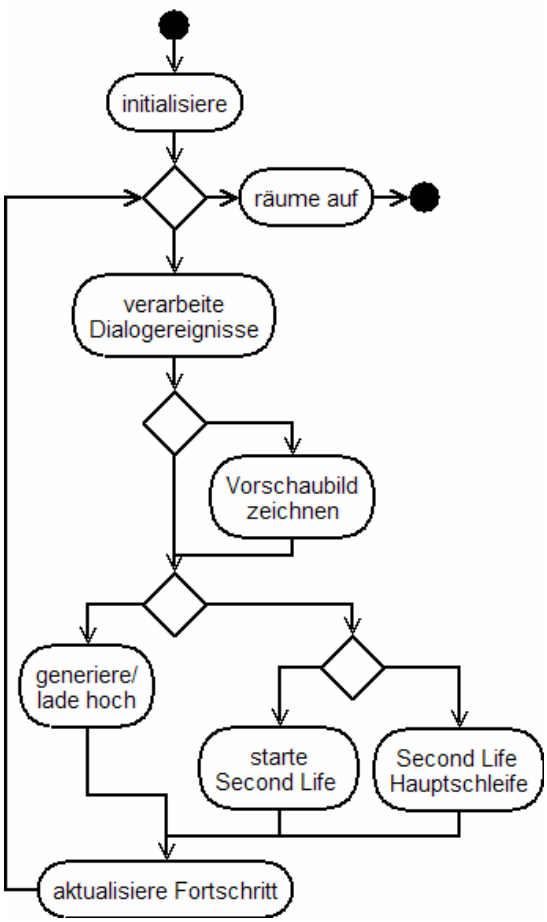

Abb. 3 Fortschrittsdialogschleife

2.4.3. Klassen

Im Folgenden sind die Aufgaben und Funktionalitäten der Klassen der „wSLGen“ Applikation beschrieben. „wSLGen“ ist die Kurzform von „weblin Second Life Generator“. Für eine Beschreibung der einzelnen Funktionen steht die Schnittstellendokumentation[38] zur Verfügung.

2.4.3.1. WGenerator

Die WGenerator Klasse ist die zentrale Klasse dieser Applikation. Sie delegiert über eigene Methoden an Second Life oder eine der drei Threadklassen[39]. Ganz zentral ist die „run“ Methode, die eine „switch“-Anweisung über die globale Statusvariable definiert und für jeden Status eine Methode aufruft. Diese Methode wird, wie schon beschrieben, aus der Hauptschleife des Hauptprogramms zum Anzeigen des Fortschrittsdialogs aufgerufen oder aber aus der „SLMainLoop“ Methode, die einen Durchlauf der Second Life Hauptschleife darstellt. Ist ein Status, z.B. das Starten der ersten Animation inklusive Setzen der Klassenvariablen auf Defaultwerte beendet, wird der Status neu gesetzt. Die Methode delegiert an die nächste Methode, die eine andere Funktionalität erfüllt. Im Beispiel würde die aufgerufene Methode alle Animationen abspielen und die Rohdaten der Bilder speichern, sowie bei jedem Renderschritt ein Vorschaubild generieren. Das Rendern selbst findet in einer Membermethode der WGenerator Klasse, der „doSnapshot“ Methode, statt. Diese Methode ruft die „display“ Methode von Second Life auf. Diese rendert ein Bild, welches danach mit der OpenGL Methode „glReadPixels“ als RGBA[40] und Alpha ausgelesen wird.

Da beim Rendern in Second Life keine Alpha- oder Transparenzwertangabe für den Hintergrund möglich ist, wird der Tiefenwert Buffers als Alpha-Wert verwendet und jeder A-Kanal-Wert durch 255 ersetzt, wenn der Tiefenwert maximal ist, also kein Objekt an dieser Stelle existiert. Ansonsten wird der Wert auf null gesetzt. Da der Second Life Client noch keine Anti-Aliasing bei Figuren unterstützt stören diese harten Kanten nicht, da sie durch das Skalieren vor der Speicherung neutralisiert werden.

---

[38] Vgl. Kapitel Schnittstellendokumentation

[39] Vgl. Kapitel 2.4.3.2.

[40] Akronym für Red Green Blue Alpha, vier Kanaldarstellung eines Pixel

Die einzelnen Methoden, die von „run“ aufgerufen werden, sind:

- „initImageGen“ initialisiert das Rendern. Sie holt beim ersten Ausführen, wenn Second Life soweit gestartet wurde, dass gerendert werden kann, die Kamera und Avatardaten, setzt den Sonnestand auf mittags, ebenso wie sie die Standard „stehen“-Animation lädt und ausführt. In den folgenden Durchläufen ruft sie die beschriebene „doSnapshot“ Methode auf, die aber keine Animationsbilder abspeichert. Es werden nur Vorschaubilder für den Fortschrittsdialog generiert, da noch keine Animationen geladen wurden und gerade ausgeführt werden. Diese Methode wird bei jedem Schleifendurchlauf aufgerufen, bis der Benutzer den Startknopf auf dem Fortschrittsdialog drückt. Dadurch wird der Status erhöht und „startImageGen“ ausgeführt.

- „startImageGen“ wird falls kein Fehler auftritt nur ein Mal ausgeführt. Sie ruft unter anderem die „loadAnimations“ Methode auf, die als Parameter das Geschlecht des Avatars bekommt und die dementsprechende XML Datei lädt („gestures.xml“ oder „gesturesFemale.xml“). Dafür wird die XML Datei geparsed und die Animation in die Liste aller Animationen eingetragen. Wenn die zu ladende Animation eine Datei und keine vordefinierte Animation ist, wird die „loadAnimFromFile“ Methode aufgerufen, die einen LLBvhLoader erzeugt und die Animation in den Cache lädt. Die Zugriffs-ID als LLUUID-Objekt wird zurückliefert, und kann somit in die Liste der Animationen mit aufgenommen werden. Nachdem die Animationen vollständig geladen sind wird die erste Animation gestartet und der Status erhöht, so dass „doImageGen“ ausgeführt wird.

- „doImageGen“ füllt die Animationen in der Animationsliste mit Bildern. Dabei werden entweder so viele Bilder gerendert, bis die Animation abgeschlossen ist oder, wenn eine feste Anzahl von Bildern für die Animation angegeben ist, wird genau diese Anzahl gerendert. Die Bilder werden in der „doSnapshot“ Methode in die Bilderliste gespeichert, wobei jeweils das letzte gerenderte Bild zusätzlich auf die pLastImage_ Variable zugewiesen wird nachdem es auf die Größe des Vorschaubildes verkleinert wurde. Dieser Schritt wird in der „initImageGen“ ebenfalls ausgeführt. Sobald eine Animation abgeschlossen ist wird der Avatar in jedem Fall wieder in die Ausgangsposition bewegt, für den Fall, dass er sich zu sehr von dieser entfernt haben sollte. Durch dieses Vorgehen startet die folgende

Animation flüssig im Verhältnis zur Ausgangsposition. Die „doImageGen“ Methode wird so lange aufgerufen, bis die letzte Animation erreicht wurde. Sie iteriert mit Hilfe der Methoden „startAnimation“, „nextAnimation“ und „stopAnimation“ über die gesamte Liste. Wenn der Laufzeiger keine aktuelle Animation mehr enthält, wird der Status gesetzt, so dass „saveImageGen“ im nächsten Schleifendurchlauf der Fortschrittsdialogsschleife ausgeführt wird.

```
//laufende Animation sofort abbrechen = true
gAgent.getAvatarObject()->stopMotion(pAnimation_->animID,true);

if(standID.notNull()){ //bewege Avatar zur Ausgansposition
  gAgent.getAvatarObject()->startMotion(standID);
}else{ //Wenn keine standard Stehen-Animation exisitert
  gAgent.getAvatarObject()->startMotion(ANIM_AGENT_STAND_1);
}

nextAnimation(); //nächste Animation starten
```

- „saveImageGen“ löscht zuerst alle existierenden Bilder, die in Ordnern liegen, die durch das Speichern mit neuen Bildern gefüllt werden, damit es nicht zu eventuellen Namenskollisionen mit anderen Animationen, die gleich benannt wurden aber mehr Bilder generiert haben, kommt. Sind die Bilder erfolgreich gelöscht worden wird der WSavePNGThread[41] gestartet und der Status erhöht um die „doSaveWait“ Methode auszuführen.

- „doSaveWait“ verzögert um eine Millisekunde, solange der WSavePNGThread ausgeführt wird. Wird vom Thread die „saveFinished“ Methode aufgerufen, die sich in der gleichen Quellcodedatei wie der WGenerator befindet, so wird je nach Modus, in dem das Programm betrieben wird, der Dialog angezeigt. Dieser besagt, dass das Erstellen erfolgreich war oder es wird mit „doImagesToXml“ fortgefahren. Der Dialog erscheint nur wenn „Erstelle Bilder“[42] ausgewählt wurde.

- „doImagesToXml“ lädt alle Animationen neu, falls diese noch nicht geladen wurden und startet den WGenXMLThread[43]. Darauf wird der Status für die „doItXWait“ Methode angepasst. „ItX“ steht hier für „ImagesToXml“.

[41] Vgl. Kapitel 2.4.3.2.2
[42] Vgl. Kapitel 6.3.3.2
[43] Vgl. Kapitel 2.4.3.2.3

- „doItXWait“ existiert nur für die Lesbarkeit des Programmcodes, da sie dieselbe Funktionalität besitzt wie „doSaveWait“ und keine Operationen ausführt. Die Verzögerung besteht solange, bis WGenXMLThread die „imageToXmlFinished“ Methode startet. Diese funktioniert analog zur „saveFinished“ Methode. Sie gibt die Liste der Animationen frei, löscht den aufrufenden Thread, erhöht den Status oder, wenn nur „Erstelle XML“[44] ausgewählt wurde, sorgt sie dafür, dass der Dialog angezeigt wird, dass das Erstellen abgeschlossen ist.

- „doUploadXml“ lädt die XML Datei, initialisiert den HTTP Client und setzt URI, Header und Body zum Verschicken der Datei an den weblin Server. Dann wird der WHTTPClientWrapper[45] gestartet und wie in den vorherigen Fällen auch der Status gesetzt, so dass „waitUploadEnd“ ausgeführt wird.

- „waitUploadEnd“ hat eine etwas andere Funktionalität, als die beiden vorherigen Methoden zum Warten auf den jeweiligen Thread. Sie aktualisiert in jedem Schritt die Fortschrittsvariable und fragt eventuelle Fehler ab. Erst wenn der Fortschritt 100 Prozent erreicht hat, wird in jedem Fall ein Dialog angezeigt, dass das Hochladen erfolgreich abgeschlossen wurde. Der Thread wird ebenfalls beendet und das Objekt entfernt.

[44] Vgl. Kapitel 6.3.3.3
[45] Vgl. Kapitel 2.4.3.2.1

2.4.3.2. Threadklassen

Threads sind sozusagen „Unterprozesse“ einer Applikation, welche aber unabhängig von dieser vom System bearbeitet werden. Somit ist es möglich eine Berechnung durchzuführen, die unter Umständen mehrere Minuten in Anspruch nimmt und nahezu zeitgleich den Anzeigedialog zu aktualisieren, damit es für den Benutzer nicht so aussieht, als ob die Applikation nicht reagieren würde. Zusätzlich können Hauptprogramm und Thread kommunizieren um z.B. in dem Dialog den Fortschritt einer Berechnung anzuzeigen während diese läuft. Für die Kommunikation müssen Funktionen oder Variable definiert werden.

Die „Thread“ Klasse erzeugt hierfür durch Aufrufen der „Start“ Methode einen Windows Thread, der wiederum die vollständig virtuelle Methode „Execute“ aufruft. Diese muss in jeder von der „Thread“ Klasse erbenden Klasse implementiert sein und in ihr muss die komplette Funktionalität des Threads abgehandelt werden. Das bedeutet bei den drei aufgeführten Klassen, dass diese wie eine normale Methode durchlaufen werden, da diese Threads nur Programmfunktionalität auslagern um den Fortschrittsdialog nicht zu blockieren.

2.4.3.2.1. WHTTPClientWrapper

Der WHTTPClientWrapper erbt von der „WHTTPClient“ Klasse. Diese wiederum erbt von der „Thread“ Klasse und der WHTTPClient besitzt dadurch zwangsweise eine „Execute“ Methode. Diese Methode führt einen kompletten HTTP-Request durch und ruft dabei die fünf vollständig virtuellen Methoden „Opened“, „Header“, „DataIn“, „Failed“ und „Closed“ auf. Diese sind in WHTTPClientWrapper zu implementieren.

„Opened“ wird aufgerufen, wenn die Verbindung geöffnet werden konnte und der HTTP-Request erfolgreich abgeschickt wurde. In diesem Fall wird im WHTTPClientWrapper nur die Fortschrittsvariable gesetzt, die vom Fortschrittsdialog über „getProgress“ ausgelesen werden kann. Ebenso wird beim Aufruf von „Closed“ verfahren. Diese Methode wird aufgerufen, wenn der HTTP-Request vollständig abgeschlossen wurde. Wenn ein HTTP-Response-Header empfangen wurde, wird die Methode „Header“ aufgerufen. Sie aktualisiert ebenso wie „DataIn“ die Fortschrittsvariable. Zusätzlich reicht sie den HTTP Statuscode und die weiteren empfangenen Header über eine Callbackfunktion, die von der „WGenerator“ Klasse über die „setHeaderCallback“ Methode gesetzt wurde an das Hauptprogramm weiter, damit die Daten nach dem Löschen des Threads nicht verloren gehen. Wie für die „Header“ Methode gilt auch für „DataIn“, dass die eingegangenen Daten des Body und deren Länge über die von der „setDataCallback“ Methode gesetzten Funktionsreferenz zurück an das Hauptprogramm gegeben werden.

2.4.3.2.2. WSavePNGThread

Diese Kindklasse der „Thread“ Klasse speichert die als unkomprimierte 500 mal 500 Pixel vorliegenden Einzelbilder aller Animationen sortiert nach der jeweiligen Animation in einzelnen Unterordnern ab. Die Bilder sind in ihrem Rohzustand noch viel zu groß, als dass es sinnvoll wäre sie hoch zu laden (ca. 340 Megabyte). Daher werden sie auf eine Größe von 100 mal 100 Pixel herunter skaliert und mit Hilfe der CxImage[46] Bibliothek beim Speichern als PNG enkodiert. Ohne weitere Bearbeitung wäre der Avatar nur klein in der Mitte zu sehen. Daher müssen die Bilder vor dem Skalieren auf genau 100 mal 100 Pixel beschnitten werden. Wichtig ist, dass alle

[46] Vgl. Kapitel 2.4.1.7

gleichermaßen bearbeitet werden, damit die Animationen flüssig abgespielt werden können. Dafür wird bei jedem Bild über alle Pixel iteriert und geprüft, ob die Alpha-Komponente ungleich null ist. Null bedeutet hier vollständig transparent. Solche Pixel können folglich vernachlässigt werden. Somit wird die absolute Ausdehnung des Avatar in allen Animationen wird ermittelt.

Wenn das Verhältnis der Bilder von Höhe zu Breite nach der Beschneidung negativ ist, wird die Breite auf 100 Pixel aufgefüllt und dabei darauf geachtet, dass die Bilder zentriert in den 100 Pixel liegen. Wenn das Verhältnis positiv ist, werden die Bilder nach oben hin aufgefüllt, da der Avatar später auf der Browserleiste stehen soll und nicht „schweben“ darf. Dies wird durch die folgenden sieben Codezeilen errechnet.

```
if(nWidth > nHeight){
  nMinHeight -= nWidth - nHeight;
  nHeight = nWidth;
} //set to width, because not normalized to 100 Pix yet
if(nWidth < nHeight){
  nMinWidth -= ((nHeight - nWidth)/2);
  nMaxWidth += ((nHeight - nWidth)/2);
  nWidth = nHeight;
}
```

Da Bilder sehr große Mengen an Speicherplatz und zum Verarbeiten unter Umständen viel Zeit benötigen wird, wird hier besonders darauf geachtet, dass alle Bildobjekte und ihre Daten wieder freigegeben werden, sowie dass keine unnötigen Operationen, wie Überprüfungen, die nicht unbedingt notwendig sind, abgearbeitet werden müssen.

Es ist leider notwendig die aus dem Grafikbuffer gelesenen RGBA Daten zu manipulieren und den roten mit dem blauen Kanal zu vertauschen, da die Bilder ansonsten von der CxImage Bibliothek als Falschfarben-Bild abgespeichert werden, was nicht gewünscht ist.

2.4.3.2.3. WGenXMLThread

WGenXMLThread ist eine normale von der „Thread“ Klasse erbende Klasse, die nicht über Zwischenstufen erbt. Sie dient, wie der Name schon andeutet, dazu eine XML Datei zu generieren. Diese XML Datei, die später als „send.xml“[47] abgelegt wird, stellt gleichzeitig eine Avatarkonfiguration dar, ebenso wie sie die Bilder der Animationen enthält. Um diese Datei zu erstellen wird der normale Baum einer Avatarkonfiguration erstellt und die einzelnen *sequence* Tags werden mit *image* Tags gefüllt, je nachdem wie viele Bilder diese Animationssequenz besitzt. Die *image* Tags sind mit den Bilddaten des jeweiligen Bildes als PNG codiert gefüllt. Dazu werden die Bilddateien, welche vom WSavePNGThread generiert wurden, binär geladen und Base64 enkodiert. Base64 verzichtet auf zwei Bits je Byte und schafft so eine Binärdarstellung, die in Textdateien abgebildet werden kann und dabei ausschließlich 64 Zeichen verwendet, die in jeder Zeichencodierung dargestellt werden können. Dafür werden die 26 Klein- und 26 Großbuchstaben, die Zahlen null bis neun und zwei zusätzliche Zeichen verwendet, die in den ersten 128 Zeichen ASCII[48] liegen und keine Steuerzeichen sind. Die als PNG abgespeicherten Bilder können durch Komprimierung nur sehr minimal in der Dateigröße optimiert werden. Die fertige XML Datei wird dennoch per gzip[49] komprimiert, da die Komprimierung die um 25 Prozent erhöhte Dateigröße, welche durch die Base64 Enkodierung entstanden ist, wieder aufhebt. Sobald dieser Vorgang abgeschlossen ist, wird die Callbackmethode ausgeführt, die das Hauptprogramm benachrichtigt, dass der Thread erfolgreich ausgeführt wurde und das Threadobjekt löscht.

[47] Vgl. Kapitel 2.4.1.3
[48] American Standard Code for Information Interchange
[49] GNU zip, Link: http://www.gzip.org/

2.4.3.3. Dialogklassen

WDialog ist eine abstrakte Klasse, die über ihre „Create“ Methode einen normalen Dialog oder über die „DoModal“ Methode einen modalen Windowsdialog erstellt. Außerdem wird die Callbackfunktion registriert über die einzelne Methoden wie „OnInitDialog“, „OnCommand“, „OnLButtonDown“ oder „OnQuit“ aufgerufen werden, wenn das entsprechende Ereignis aufgetreten ist. Diese Methoden sind bereits alle in der abstrakten Klasse implementiert, besitzen aber keine Funktionalität. Das Benutzerinterface verwendet zur Darstellung ausschließlich WinAPI[50] Methoden.

2.4.3.3.1. WLoginDialog

Der WLoginDialog besitzt keine besondere Funktionalität, da hier nur das Drücken der Knöpfe Ereignisse auslöst, die verarbeitet werden. Wenn ein anderer Knopf als „Erstelle XML“ gedrückt wurde, werden die Benutzereingaben in den Dialogfeldern auf Existenz überprüft. Die Prüfung auf die Richtigkeit der angegebenen Daten wird anderen Programmteilen, bzw. den Autorisationsskripten auf den Servern, überlassen.

2.4.3.3.2. WProgressDialog

Für den WProgressDialog ist der größte Teil des Aufwands die Aktivierung und Deaktivierung der Dialogelemente, die in der „initMode“ Methode bei jedem Wechsel vom WLoginDialog gesetzt werden. Besonders hervorzuheben ist die „OnRedraw“ Methode, die das Vorschaubild zeichnet, vorausgesetzt Second Life wurde gestartet und es existiert eine aktive Bildreferenz auf ein LLImageRaw im „glob::pLastImage_“. Dieses Bild wird über mehrere Windows Funktionen in den Buffer eines OpenGL-Fensters kopiert, welches auf dem Dialog angezeigt wird. Hier wird sehr darauf geachtet, dass alle Bildressourcen wieder freigegeben werden, da diese einige GDI-Objekte benötigen, die wenn sie nicht wieder freigegeben werden an dieser Stelle innerhalb von Sekunden dazu führen können, dass keine Applikation auf dem ausführenden Rechner mehr fehlerfrei läuft.

[50] Windows Application Programming Interface

2.4.3.3.3. WChooseLangDialog

Dieser sehr einfache Dialog bietet eine aufklappbare Auswahlbox, die beim Initialisieren des Dialogs mit den verfügbaren Sprachen gefüllt wird. Der Dialog kann nur modal aufgerufen werden, es wird also die restliche Applikation blockiert solange dieser Dialog angezeigt wird. Der Vorteil dabei ist, dass keine Schleife benötigt wird, die die Ereignisse verarbeitet, sondern die Methode auf das Beenden des Dialogs wartet und die Auswahl, die in der Auswahlbox getroffen wurde, verarbeitet. Die aufrufende Funktion[51] kann die Auswahl so weiterarbeiten.

2.4.3.4. Hilfsklassen

Die Hilfsklassen sind kleine Klassen die Funktionalität auslagern um sie einfach in mehreren anderen Klassen verwenden zu können. Dabei sind die einzelnen Methoden anhand ihrer Funktionalität in die einzelnen Klassen unterteilt. Alle Hilfsklassen fungieren nach dem Singleton-Muster, d.h. es gibt im Speicher nicht mehr als eine Instanz dieser Klasse, da alle Methoden und Variablen der jeweiligen Klasse statisch sind und somit nur einmalig beim Starten der Applikation geladen werden.

2.4.3.4.1. BoxDialog

Diese Klasse könnte den Dialogklassen zugeordnet werden, sofern sie eigenständige Windowsdialoge erzeugen würde. Die Boxdialog-Klasse kapselt in diesem Sinne die WinAPI-Funktion MessageBox um verschiedene Arten von Dialogen zu erzeugen ohne die Parameter für die Art der MessageBox angeben zu müssen, wie z.B. welche Buttons angezeigt werden sollen und welches Symbol der Dialog haben soll.

2.4.3.4.2. File

Die File-Klasse bietet einige Basisdateioperationen, wie das Überprüfen ob eine Datei mit gegebenem Namen existiert oder das Ermitteln der Größe einer geladenen Datei. Zusätzlich bietet sie die Möglichkeit eine Textdatei mit Schlüssel-Wert-Paaren in eine Liste aus solchen Paaren zu parsen. Es ist dabei möglich eine komplette Datei oder einzelne Zeilen zu verarbeiten, wobei bei der Datei darauf zu

[51] Vgl. Kapitel 2.4.3.4.5

achten ist, dass die einzelnen Paare durch Zeilenumbrüche oder einfache Absätze (Carriage return / line feet) getrennt sind. Die Werte der Schlüssel-Wert-Paare werden zudem noch auf zu ersetzende Zeichenketten untersucht. So wird z.B. aus „[[BR]]“ in der Textdatei ein Zeilenumbruch („\n“) in der verarbeiteten Zeichenkette, da in der Datei keine Zeilenumbrüche im Schlüssel oder Wert enthalten darf, wie in den Dateistrukturen[52] beschrieben.

2.4.3.4.3. glob

Die Klasse glob hält Objekte vor, die an verschiedenen Stellen in der Applikation benötigt werden. Da diese Stellen teilweise sehr tief in der Klassenhierarchie von Second Life liegen oder an anderen Stellen sehr viele der Variablen benötigt werden, sind diese Variablen global verfügbar. Dies führt dazu, dass die Schnittstellen klein gehalten werden und z.B. auf das „OpenGL“-Fenster in welches Second Life rendert direkt zugegriffen werden kann. Zusätzlich dient die Klasse dazu, die Dialogfenster mit Informationen zu versorgen, da diese unabhängig von der Datenverarbeitung behandelt werden, wechselseitig aber Einfluss nehmen können. Dieser Einfluss beschränkt sich vom Dialog aus auf Textfeldeingaben und dem Aktivieren von Knöpfen.

2.4.3.4.4. Ini

Die Ini-Klasse wird benötigt um die „config.ini“[53] Datei zu laden oder, falls diese nicht existiert, sie zu erstellen und mit Standardwerten zu füllen.

2.4.3.4.5. Lang

Wie der Klassenname schon andeutet koordiniert die Lang-Klasse die Steuerung der Texte in der jeweiligen Sprache. Dafür wird beim Programmstart die Initialisierungsmethode dieser Klasse aufgerufen um die entsprechende Sprachdatei zu laden. Schlägt das Laden fehl, wird versucht die Sprachdatei für Englisch zu laden („en.lang“). Schlägt dies ebenfalls fehl, wird das Programm trotzdem gestartet. Alle zu übersetzenden Texte werden in diesem Fall durch „Unknown translation“ ersetzt. Wenn eine Sprachdatei erfolgreich geladen wurde kann durch die Methode „Translate“ und die Angabe des Schlüssels der Text in der richtigen Sprache

[52] Vgl. Kapitel 2.4.1.3
[53] Vgl. Kapitel 2.4.1.3

gefunden und somit im Dialog angezeigt werden. Die Schlüssel müssen dementsprechend in jeder Sprachdatei[54] gleich sein.

2.4.3.4.6. Tools

Tools enthält Methoden, die keiner anderen Klasse eindeutig zuzuordnen sind, wie z.B. das Erstellen von Teilen von Dateipfaden für die zu speichernden Bilder, die genau aus einer vierstelligen Nummer bestehen sollen und somit mit Nullen von vorne aufgefüllt werden müssen. Des Weiteren ist das Setzen von Dialogtexten je nach Status der Applikation enthalten.

2.4.3.5. Wichtige verwendete Second Life Klassen

Die unter diesem Punkt aufgeführten Klassen gehören zu Second Life, werden direkt oder indirekt von den erstellten Klassen verwendet und nehmen eine zentrale Position im Programmablauf ein.

2.4.3.5.1. LLAgent

LLAgent ist die Repräsentation des Clients in einem Objekt. Über das in Second Life global definierte Objekt „gAgent“ kann von überall auf den eigenen Avatar und ähnliches zugegriffen werden. Beispielsweise kann indirekt über das LLVOAvatar Objekt des eigenen Avatars auf das Geschlecht zugegriffen werden, um die entsprechende Animationsdefinitionsdatei zu laden. Im Folgenden werden einige Codebeispiele für Zugriffe aufgeführt:

```
//erstellt eine neue Animation im Cache
gAgent.getAvatarObject()->createMotion(mMotionID);

//startet eine (erstellte) Animation anhand der ID
gAgent.getAvatarObject()->startMotion(standID);

//gibt die globale Position des Charakters zurück für die
//Kameraausrichtung
gAgent.getAvatarObject()->getCharacterPosition();

//gibt das Geschlecht des Avatars zurück
gAgent.getAvatarObject()->getSex();
```

[54] Vgl. Kapitel 2.4.1.3

2.4.3.5.2. LLBVHLoader

Der LLBVHLoader wird verwendet um BVH[55] Dateien zu laden und die Animationen in den Second Life Cache zu laden. Währenddessen ermöglicht er mit Hilfe einer LLTransactionID die zugehörige LLUUID der Animation zu ermitteln.

2.4.3.5.3. LLKeyframeMotion

LLKeyframeMotion verarbeitet die Animationsdaten aus dem Cache und wendet sie auf die einzelnen Gelenke des Avatars an. Wenn Zwischenschritte nötig sind, wird interpoliert. Da die Animationen aber meist für zehn Frames pro Sekunde ausgelegt ist und der Second Life interne Timer so umgeschrieben wurde, dass er die Animation simuliert, dass alle 0,1 Sekunden ein Animationsschritt gerendert wird, ist es meistens nicht nötig die Animationen zu interpolieren.

2.4.3.5.4. LLImageRaw

LLImageRaw Objekte stellen unkomprimierte Bilder dar, die einige Basisfunktionalitäten besitzen. Sie können skalieren, ausschneiden, Höhe, Breite und Größe auslesen oder Zeiger auf die eigentlichen Bilddaten erhalten um diese zu bearbeiten. Diese Klasse wird verwendet um alle generierten Bilder zu speichern, ob für die Vorschau oder um sie später zu den finalen Einzelbildern der Animationen zu bearbeiten.

2.4.3.5.5. LLUUID

LLUUIDs sind Zeichenketten aus Nummernpaaren, die durch Minus verbunden sind. UUID[56] steht für „Universal Unique Identifier“. Also eindeutige Identifikationsnummern, die für etliche Zwecke verwendet werden, beispielsweise um eine Textur, ein Objekt, einen Avatar, eine Animation oder ähnliches zu identifizieren. Über diese IDs können aus dem virtuellen Dateisystem von Second Life oder seinem Cache Daten auf Anfrage erhalten werden, die eindeutig zuzuordnen sind.

[55] Bounding Volume Hierarchies
[56] Universal Unique Identifier

2.4.3.5.6. LLViewerCamera

Die LLViewerCamera bestimmt unter anderem Blickrichtung und Sichtweite auf den Avatar. Daher ist sie sehr wichtig, weil ihr „point of view“ immer genau auf die Mitte des Avatars gerichtet sein sollte und sie nicht zu weit weg von dem Avatar sein darf, so dass Objekte nicht mehr gezeichnet werden. Die Entfernung, in der Objekte noch gezeichnet werden, darf aber nicht zu groß sein, da es bei Objekten in größerer Entfernung vorkommen kann, dass diese noch keine Zuweisung zu einer Gruppe haben und deshalb durch die Änderungen in der „display“ Methode nicht gefiltert werden. Desto weiter die Objekte weg sind, desto länger wird es dauern, bis die Gruppenzuordnung empfangen und das Objekt vom Zeichnen ausgeschlossen wird. Ebenso wird die Kamera benötigt um die „walk“ Animationen zu rendern, da auf die Kamera keine physikalischen Kräfte einwirken und der Avatar sich flüssig bewegen soll.

2.4.3.5.7. LLViewerObjectList

Die LLViewerObjectList verwaltet alle Objekte und ist somit der ideale Ansatzpunkt, Objekte, die nicht zum eigenen Avatar gehören, nicht anzuzeigen. Löschen ist hier leider nicht möglich, da nur Löschaktionen möglich sind, die dem Server sofort mitgeteilt werden. Der Benutzer erhält daraufhin unnötig viele Nachrichten, wenn versucht wird Objekte zu löschen, die dieser in der Second Life Welt nicht besitzt. Daher werden die Objekte mit null skaliert und sind somit auch nicht mehr visualisierbar.

```
LLVOAvatar* av = LLVOAvatar::findAvatarFromAttachment(objectp);
//wenn der Avatar zu dem Objekt nicht der eigene ist
if(av && !av->mIsSelf){
  av->mVisible= false;
  objectp->setScale(LLVector3(0.f,0.f,0.f));
}
```

2.4.4. Wichtige Änderungen an Second Life Dateien

- „viewer.cpp“: Die „WinMain“ und die „main_loop“ Methoden werden zu den Methoden „SLMainStart“, „SLMainLoop“ und „SLMainEnd“. „SLMainStart“ führt einen normalen Programmstart von Second Life aus ohne dabei die Flashscreens anzuzeigen und initialisiert dabei die „SLMainLoop“. Diese führt einen einzigen Durchlauf der früheren „main_loop“ durch die um das Handling von Benutzereingaben und Screenshots erleichtert wurde. Die „SLMainEnd Methode führt nur den Teil aus, der in der früheren „WinMain“ Methode hinter dem Aufruf von „main_loop“ stand und räumt somit Second Life wieder auf.

- „pipeline.cpp“: unterstützt die „llviewerdisplay.cpp“, die „lldrawpoolsimple.cpp“ und die „lldrawpoolavatar.cpp“ indem der Renderingvorgang ebenfalls durch Weglassen von Methodenaufrufen erleichtert wird. Außerdem wird eine Fallentscheidung modifiziert, so dass nur Avatare und einfache Objekte gerendert werden.

```
//nur zulassen von Avatare und einfachen Objekten
if(hasRenderType(poolp->getType()) && poolp->getNumPasses() > 0 &&
  (poolp->getType() == LLRenderTypeMask::RENDER_TYPE_AVATAR ||
   poolp->getType() == LLRenderTypeMask::RENDER_TYPE_SIMPLE))

...
// Ausschluss der Gegenstandsgruppe 6 mit der ID 3, die nicht
// benötigt werden
// Gruppe 5 und 7 sind Avatare und benötigte einfache Objekte
    if(!((poolp->getId() == 3) && (p->getType() == 6))){
      if (p->getType() != cur_type)
        break;
      p->render(i);
    }
```

- „llvoavatar.cpp“: Normale Bewegungen, wie z.B. blinzeln, werden bei der Animation unterdrückt, da diese auch zu ungewollten Bewegungen des Körpers führen können, die in den Animationen später Ruckeln verursachen können.

- „llviewerwindow.cpp“: Alle HUD[57]-Elemente werden nicht in die Liste der zu zeichnenden Elemente aufgenommen, so dass keine Menüs sichtbar sind.

[57] Head-Up-Display

- „llviewerobjectlist.cpp“: Vgl. Kapitel 2.4.3.5.7

- „llviewerdisplay.cpp“: Die „display“ Methode, die in der „doSnapshot“ Methode des WGenerator verwendet wird, wird so modifiziert, dass während des Ladens nichts angezeigt wird und ist weitestgehend optimiert, so dass keine unbenötigten Elemente, wie HUD-Elemente mehr angezeigt werden.

- „llviewercamera.cpp“: Hier wurde die Sichtweite der Kamera angepasst. Sie wurde von fünfzig auf vier Meter gesetzt.

- „llstartup.cpp“: Das Anzeigen des Second Life Anmeldedialogs wird übergangen und die Daten, die auf dem WLoginDialog eingegeben wurden, werden anstatt derer, die ehemals über den Anmeldedialog eingegeben wurden, an den Server geschickt. Dazu muss das Passwort MD5[58] kodiert und zum Hexadezimalwert konvertiert werden.

```
char hexpass[MD5HEX_STR_SIZE];
LLMD5 passmd5;
passmd5.update((unsigned char*)glob::sSLPass_.c_str(),
                               glob::sSLPass_.bytes());
passmd5.finalize();
passmd5.hex_digest(hexpass);

gUserAuthp->authenticate(
  auth_uris[auth_uri_num].c_str(),
  auth_method.c_str(),
  glob::sSLFirst_.c_str(),
  glob::sSLLast_.c_str(),
  hexpass, ...
```

- „llpreviewanim.cpp“: Alle Animationen werden nur lokal abgespielt, die Daten werden nicht an den Second Life Server geschickt.

- „lldrawpoolsimple.cpp“: Das Rendern des Leuchtens von Gegenständen ist entfernt und das Rendern der Gegenstände wird auf die aktiven durch Auskommentieren der Anderen beschränkt.

[58] Message Digest Algorithm 5, Hashfunktion zum Verschlüsseln von Daten

- „lldrawpoolbump.cpp“, „lldrawpoolground.cpp“, „lldrawpoolsky.cpp“, „lldrawpoolterrain.cpp“, „lldrawpooltree.cpp“, „lldrawpoolwater.cpp“: In diesen Dateien sind für die Klassen die „render“ Methoden entfernt worden, da solche Objekte nicht benötigt werden.

- „lldrawpoolavatar.cpp“: Das Rendern der Schatten ist auskommentiert worden, da der Avatar sonst über der Browserleiste schweben könnte, weil der Schatten mit in die Berechnung des Schnittrahmens fallen würde. Außerdem ist beim Rendern die Überprüfung, ob der zu rendernde Avatar wirklich gerendert werden soll so weit eingeschränkt, dass diese nur bei dem eigenen Avatar zutrifft.

```
//diese Überprüfung schließt alle nicht nicht eigenen Avatare vom
//Rendervorgan aus
if( (!single_avatar && avatarp->mIsSelf) || (avatarp ==
single_avatar) )
```

- „lldrawable.cpp“: Bei der Überprüfung auf Sichtbarkeit des Objektes werden alle Objekte ausgeschlossen, die nicht zu dem eigenen Avatar gehören.

```
LLVOAvatar *av =
 LLVOAvatar::findAvatarFromAttachment((LLViewerObject*)getVObj());
if(!(av && av->isSelf())){ return FALSE; }
```

- „llwindowwin32.cpp“: Erstellen der OpenGL-Fenster für das Rendern und das Vorschaubild. Weiterleiten der MessageBox Nachrichten an die erstellte, Second Life umfassende, Applikation und Unterbinden der Anzeige des Logoscreens beim Laden vom Second Life.

- „lllfsthread.cpp“, „llvfsthread.cpp“: Veränderung des Destruktors, so dass dieser sicher ist und keine Speicherzugriffsfehler mehr verursachen kann.

- „lldir_win32.cpp“: Wenn Second Life gestartet wird und im Ordner des Benutzers noch keine „settings.xml“ existiert, wird diese angelegt.

- „llfloater.cpp“, „llmodaldialog.cpp“, „llview.cpp“: Die Zeichenmethoden wurden umgangen, da die erzeugten Dialoge den Avatar überdecken würden, wenn sie auftreten. Die Mitteilungen die im LLModalDialog angezeigt würden, werden direkt an den Fortschrittsdialog weitergeleitet.

- „frametimer.cpp“: Der Applikationstimer von Second Life wurde so manipuliert, dass Second Life suggeriert wird es würde alle 0,1 Sekunden aufgerufen. Dies ist nötig um eine bestimmte Anzahl an Bildern pro Animation vorherzubestimmen und diese auch zu generieren. Dadurch läuft das Rendern der Avatare auf unterschiedlichen Maschinen natürlich unterschiedlich schnell. Es besteht aber nicht der Anspruch von Echtzeit in dem Rendervorgang.

```
//total_time ist in Mikrosekunden angegeben 100000 entspricht 0.1
//Sekunden
U64 total_time = sStartTotalTime + sFrameCount * 100000;
```

- „error.cpp“: In der Log::flush Methode wurde der Aufruf der „crashFunction“ verhindert und stattdessen das Programm normal beendet ohne gezielt einen Speicherzugriffsfehler zu verursachen. Die „crashAndLoop“ Methode des Second Life Clients:

```
int* crash = NULL;

*crash = 0;

while(true)
{
  // Loop forever, in case the crash didn't work?
}
```

2.4.5. Fehlerbehandlung

Generell wird versucht Fehler von vornherein zu vermeiden. Diese sind aber bei Benutzereingaben, verwendeten anderen Softwarestücken und erst recht bei der Kommunikation über ein Netzwerk nicht immer zu vermeiden.

Da das entwickelte Programm auf dem Second Life Client 1.18.2(1) basiert und die Oberfläche eine komplett andere ist, ist es nötig alle im Second Life Client behandelten Fehler an die neue Oberfläche weiter zu leiten, um diese dem Benutzer mitzuteilen. Auf Fehler in Second Life kann kaum Einfluss genommen werden, daher werden Fehler innerhalb des Second Life Clients nur soweit beseitigt, wie sie auftreten, reproduzierbar und lokalisierbar sind. Ebenso gibt es zu diesen Fehlern[59] keine genaueren Behebungsansätze im Benutzerhandbuch.

Fehler in neu erstellten Programmteilen werden durch verschiedene Kontrollen abgefangen und bei Auftreten wird der Variable „sError_“ der Klasse „glob“ ein Fehlertext als Inhalt gesetzt. Die Methoden besitzen, wenn Fehler auftreten können, eine bool’sche Variable „bOk“. Diese wird im Fehlerfall auf „false“ gesetzt und alle nachfolgenden Teile der Methode werden nicht mehr ausgeführt. Meistens wird im Fehlerfall zusätzlich die „bLeaveDialog_“ oder die „bQuit_“ Variable der „glob“ Klasse gesetzt. Ist der Programmablauf wieder in der Hauptschleife, z.B. vom Fortschrittsdialog angekommen, wird die Variable „sError_“ überprüft, ob sie Fehlermeldungen enthält. Ist dies der Fall werden diese über ein Dialogfenster ausgegeben. Wenn es sich nicht um kritische Fehler handelt, wird nur „bLeaveDialog_“ auf „true“ gesetzt und der Fortschrittsdialog verlassen, um wieder zum Anmeldedialog zurückzukehren. Trat hingegen ein schwerwiegender Fehler auf und die „bQuit_“ Variable besitzt den Wahrheitswert „true“, wird versucht das Programm so weit wie möglich aufzuräumen und ordnungsgemäß zu beenden.

Spezielle Fehlerklassen z.B. für Ausnahmen existieren nicht, da die Fehler alle auf die oben beschriebene Weise behandelt werden und somit keine globalen berechneten Sprünge notwendig sind.

[59] Vgl. Kapitel 6.4

3. Anwendungsfälle

Für das erstellte Programm existieren zwei mögliche Anwendungsszenarien, ebenso wie eine Kombination aus beiden.

Der erste Fall ist, der Benutzer ist in der virtuellen Welt von Second Life sehr aktiv und möchte seinen Avatar auch nutzen, wenn er sich auf Webseiten. Dies kann diverse Gründe haben. Vielleicht möchte er Andere dazu animieren bei Second Life mitzumachen, ihnen auf seiner eigenen Webseite Tipps zu Second Life geben oder Werbung für Gegenstände, Leistungen oder Skripte machen, die er in Second Life besitzt. Oder aber er möchte einfach nur zeigen wie schön und einmalig sein Avatar ist. Um dies zu realisieren kann sich der Benutzer nun einen weblin Konto anlegen und seinen Avatar aus Second Life in weblin mit verwenden. Wenn sich der Avatar in Second Life verändert, kann ein neuer generiert werden. Im weiteren Verlauf kann dieser verwendet oder weiterhin der alte benutzt werden.

Der andere Fall ist, dass der Benutzer bereits bei weblin angemeldet und dort sehr aktiv ist, sich aber einen vollständig eigenen Avatar wünscht, der ihm vielleicht sogar möglichst ähnlich sieht. Second Life bietet für einen Avatar enorm viele Einstellungen um diesen zu modellieren und auszustatten. Daher kann sich der Benutzer hier nach eigenen Wünschen einen komplett einmaligen Avatar generieren ohne Kenntnis von 3D Bearbeitungsprogrammen oder ähnlichem zu haben. Die bisherigen Mittel um einen eigenen Avatar für weblin erstellen zu lassen sind zwar sehr einfach bedienbar, aber nicht im gewünschten Maße individualisierbar, wie dies in Second Life der Fall ist. Wenn dem Benutzer der erstellte Avatar nicht mehr zusagen sollte, kann dieser einfach durch Änderung des Second Life Avatar angepasst und neu für weblin generiert werden.

Natürlich ist ebenfalls möglich, dass jemand in weblin und Second Life aktiv ist und die beiden oben aufgeführten Fälle verknüpfen will um sich einheitlich nach außen in die virtuellen Welten zu präsentieren.

4. Zusammenfassung

Die Umsetzung der Aufgabenstellung als Plugin für weblin war nicht möglich, da der Umfang und die Größe für ein Plugin zu umfangreich waren. Als eigenständige Applikation wurde das entwickelte Programm allerdings seiner Aufgabenstellung gerecht, zu der noch serverseitig im Produktivbetrieb die Implementierung der Schnittstelle zum Hochladen der Daten erfolgen muss.

Mit der erstellten Applikation kann ein Benutzer auf einfache Weise per Knopfdruck seinen Avatar von Second Life nach weblin konvertieren und dort verwenden.

Die direkte Anzeige von 3D Avataren aus nahezu beliebigen virtuellen Welten ist zurzeit mit einem unverhältnismäßig hohen Aufwand verbunden. Um die direkte Anzeige effizient nutzen zu können müssen sich erst die gerade in der Entwicklung befindlichen Standards durchsetzen, so dass nicht für jede virtuelle Welt ein eigenständiges Programm oder Plugin, welches auf dem Nutzercomputer vorinstallierte Programmteile nutzt, benötigt wird.

Wobei der größte Teil des Aufwands für die entwickelte Applikation die Einarbeitung in den Quellcode des Second Life Clients, die Kommunikation mit dem Second Life Server und die WinAPI war.

5. Ausblick

5.1. weblin Second Life Generator

5.1.1. Mögliche Erweiterungen

- Die XML Datei für Gestiken könnte z.B. um einen Status „sitzen“ erweitert werden. Für diese wären allerdings sehr viele Übergangsanimationen zu allen im stehen ausgeführten Animationen nötig, was auch die XML Datei enorm vergrößern würde.
- Die Eingabefelder können vor dem Starten einer spezifischeren Überprüfung unterzogen werden, z.B. durch Anfragen am Server, ob das eingegebene Konto existiert.
- Die Dialoge für das ausschließliche Erstellen von XML oder des Hochladens können angepasst werden, da die Freiräume auf diesen Dialogen sehr groß sind.

5.1.2. Ansatzpunkte für Optimierungen

- Die Berechnung für die Ermittlung der optimalen Bildausmaße aus dem Originalbild kann optimiert werden, auch wenn der Avatar nahezu beliebiger Struktur sein kann, ist es nicht notwendig jeden Pixel zu überprüfen.
- Die Variablen, die von Threadaufrufen oder indirekt durch deren Callbacks verwendet werden können durch einsetzen von Semaphoren komplett threadsicher gemacht werden.

5.2. Virtuelle Welten allgemein

Bei den bestehenden virtuellen Welten drängen, besonders seit Second Life so stark in den Medien präsent wurde, immer mehr neue Welten in den Markt, die völlig neue Aspekte abdecken. Wie anfangs schon beschrieben, haben sich Second Life und IBM zusammengetan um Standards zu entwickeln, die das Reisen zwischen virtuellen Welten ermöglichen. Voraussichtlich wird es in naher Zukunft Gruppen von virtuellen Welten geben zwischen denen sich ein Benutzer mit seinem Avatar nahezu frei bewegen kann. Es wird zu dieser Gruppenbildung kommen, da die Standards nicht durch eine unabhängige Instanz in den Markt eingebracht werden, sondern von den Marktführern. IBM beschäftigt mittlerweile mehr als 5000 Mitarbeiter in virtuellen Welten[60]. Die Bedeutung von virtuellen Welten wird im Allgemeinen zunehmen. Es wird dort mit Freunden gespielt, eingekauft oder sich einfach nur unterhalten. Und zwar in wesentlich weiterem Umfang als es bisher der Fall ist, da die virtuellen Welten das „Internet im Internet“ werden können, die ebenso rapide zunehmen, wie das Internet im letzten Jahrzehnt gewachsen ist.

5.3. Virtuelle Präsenzen allgemein

Virtuelle Präsenzen sind im Vergleich zu virtuellen Welten oder Webseiten noch relativ unbekannt. Sie werden aber zunehmend bekannter, da viele Benutzer nicht nur „surfen“ wollen, sondern auch kommunizieren oder Hilfe suchen. Diese erhalten sie leichter, wenn sich eine Person dort auf der Seite aufhält, die firm in der auf der Seite dargestellten Materie ist. Für virtuelle Präsenzen ist eine enorme Infrastruktur notwendig, die auf offenen Standards und dezentralisiert, nicht wie bei bekannten Instant Messagern, funktionalisieren muss. Ebenso wie das Internet selbst aufgebaut ist. Dabei wird es nötig sein die diversen Clients nicht von einem Protokoll abhängig zu machen, so dass diverse Server verwendet werden können.

[60] Vgl. http://www.heise.de/newsticker/meldung/96688, Abgerufen 2008-02-18

5.4. weblin und Second Life

Wenn die erwähnten Standards von Linden Labs und IBM durchgesetzt werden und es möglich ist direkt auf Modell und Grafikdaten von den Servern zuzugreifen, ist in Zukunft eine barrierefreie Nutzung von Avataren möglich.

Weblin wie auch Second Life werden dabei parallel ihre Systemunabhängigkeit erweitern. Weblin wird auf die zweite mögliche Aktionsdimension ausgedehnt und die Avatare werden durch Verwendung von APNG[61] weichere Kanten aufweisen. Second Life hat mit dem Windlight[62] Client schon einen Schritt in die Richtung von grafisch aufwändigeren und realistischeren Darstellungen getan. Diese bezieht sich aber nur auf Shader die z.B. Wasser, Nebel oder Leuchteffekte besser darstellen. Besonders wichtig ist hier Anti-Aliasing bei Avataren, deren Attachments und Objekten, wie Gebäude oder Bäume, um nicht ins Hintertreffen im Verhältnis zu den diversen neu entstandenen virtuellen Welten zu geraten.

[61] Animated PNG

[62] Link: http://www.second-life-info.de/sl/endlich-realistische-grafiken-in-secondlife/

6. Benutzerhandbuch

6.1. Anforderungen

Die Anforderungen entsprechen grundlegend denen des Second Life Clients[63]. Dieses Programm ist nicht unter „Mac OS“ oder „Linux“ lauffähig und baut auf dem Second Life Client 1.18.2(1) auf. Zusätzlich wird durch die Bilder mehr Arbeitsspeicher benötigt.

Dementsprechend liegen die Anforderungen bei:

Komponente	**Anforderungen**
Internetanschluss	DSL oder Kabel
Betriebssystem	Windows 2000, XP oder Vista
Prozessor	1400 MHz Pentium IV, Athlon oder besser
Arbeitsspeicher	768 MB
Grafikkarte (2000 / XP)	NVIDIA GeForce 4 oder besser, ATI Radeon 9250 oder besser, Intel 945 Chipsatz
Grafikkarte (Vista) (neueste Treiber nötig)	NVIDIA GeForce 6600 oder besser, ATI Radeon 9500 oder besser, Intel 945 Chipsatz
Festplattenplatz	300 MB, davon 230 MB auf der Systempartition

Tab. 4 Systemanforderungen des Programms

6.2. Installation

Starten Sie die „wSLGen_setup.exe“ und folgen Sie den Installationsanweisungen. Sie sollten darauf achten die nötigen Rechte auf dem Betriebssystem zu besitzen, sowie dass der Programmordner, untergeordnete Ordner und der übergeordnete Ordner nicht schreibgeschützt sind. Wenn die Installation erfolgreich abgeschlossen wurde, können Sie „wSLGen“ sofort starten oder dies manuell[64] machen.

[63] Systemanforderungen des Second Life Clients, Link: http://secondlife.com/corporate/sysreqs.php
[64] Vgl. Kapitel 6.3.3.16.3.1

6.3. Bedienungsanleitung

6.3.1. Programmstart

Um das Programm zu starten klicken Sie auf den Start-Knopf in ihrer Taskleiste, auf Programme und wählen Sie dann den Ordner "wSLGen" aus. Klicken Sie auf "wSLGen" um das Programm zu starten.

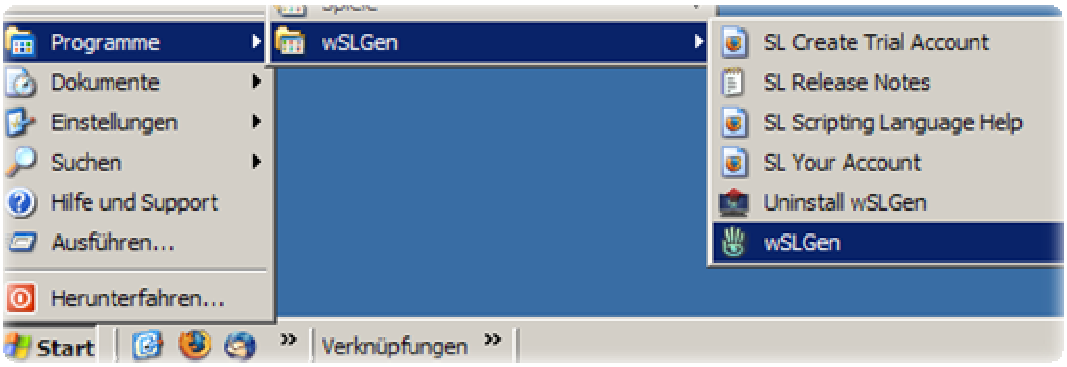

Abb. 4 Programmstart

Sie können das Programm auch direkt aus dem Installationsordner ausführen. Wenn dieser zum Beispiel "C:\Programme\wSLGen\" ist, gehen Sie in diesen Ordner und starten die "wSLGen.exe".

6.3.2. Dialoge

6.3.2.1. Anmeldedialog

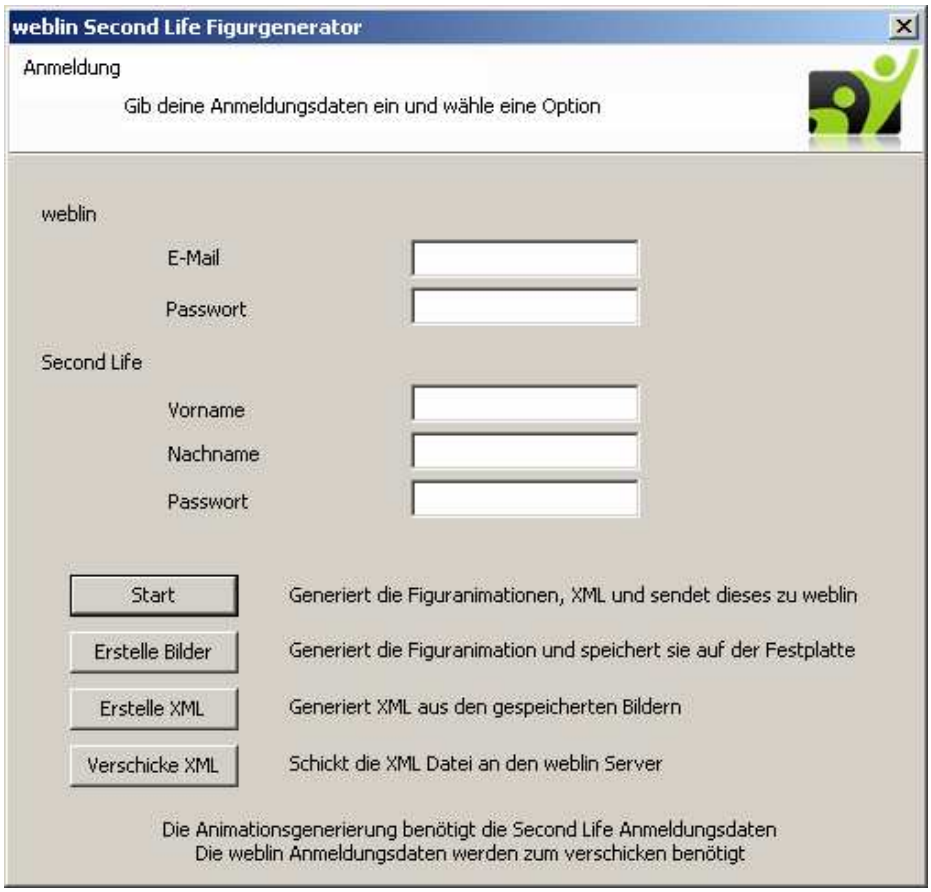

Abb. 5 Anmeldedialog

Der Anmeldedialog besteht aus fünf Eingabefeldern und vier Knöpfen. Die ersten beiden Eingabefelder sind für die Zugangsdaten zu Ihrem weblin-Account. Dafür benötigen Sie die kostenlose weblin-Vollversion (mit der Trial-Version ist dies nicht möglich). Diese Felder sind nur für die Optionen "Start" und "Verschicke XML" notwendig, da nur unter diesen beiden Optionen eine Verbindung zum weblin Server aufgebaut wird. In dem ersten Feld tragen Sie ihre E-Mail Adresse ein mit der Sie bei weblin angemeldet sind, in das zweite Feld das dazugehörige Passwort. Die drei Felder darunter sind für Ihre Second Life Anmeldungsdaten. In das erste Feld geben Sie den Vor- in das zweite den Nachnamen ihrer Figur ein und in das dritte Feld das dazugehörige Passwort. Diese Felder werden nur für die Optionen "Start" und "Erstelle Bilder" benötigt.

6.3.2.2. Fortschrittsdialog

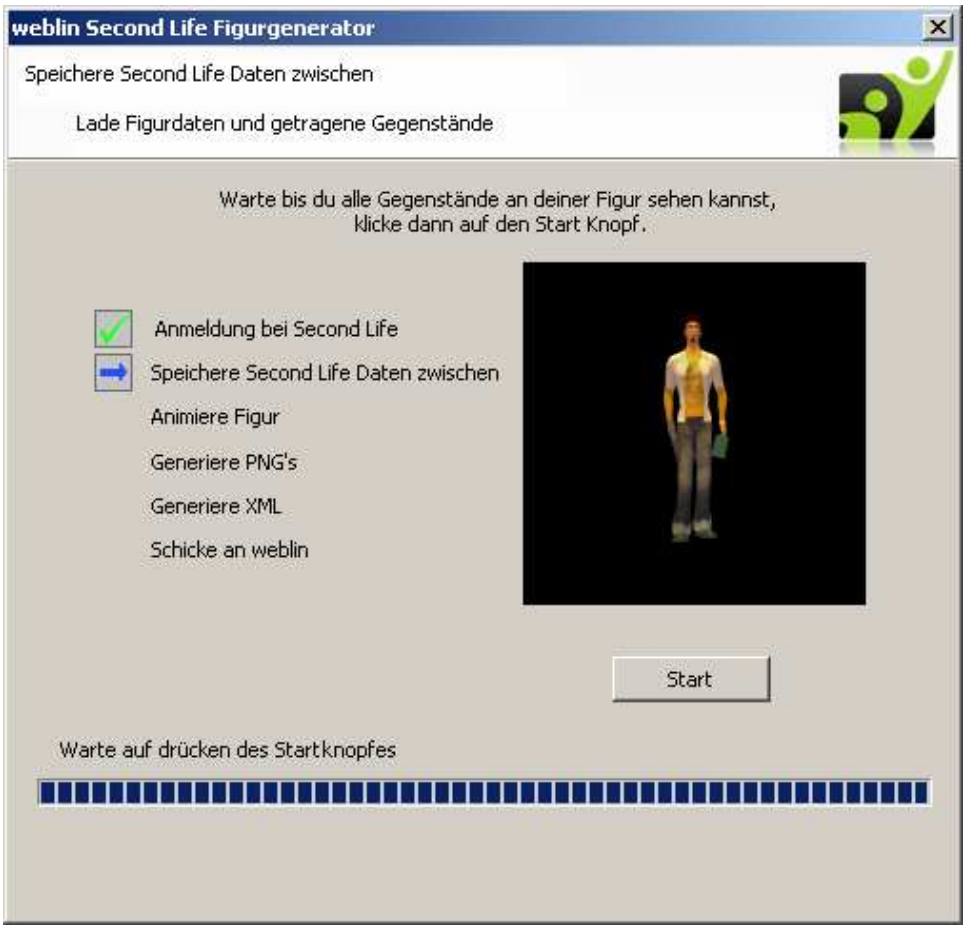
weblin Second Life Figurgenerator
Speichere Second Life Daten zwischen
Lade Figurdaten und getragene Gegenstände
Warte bis du alle Gegenstände an deiner Figur sehen kannst,
klicke dann auf den Start Knopf.
Anmeldung bei Second Life
Speichere Second Life Daten zwischen
Animiere Figur
Generiere PNG's
Generiere XML
Schicke an weblin
Start
Warte auf drücken des Startknopfes

Abb. 6 Fortschrittsdialog

Der Fortschrittsdialog allgemein besteht aus einer Kopfzeile, in der textuell die momentane Aktion und eine Kurzbeschreibung dazu festgehalten werden. Mittig auf der linken Seite ist eine Übersicht über die zu erledigenden und erledigten Aktionen aufgeführt. Auf der rechten Seite ist ggf. ein Vorschaufenster zu sehen in dem Ihre Figur angezeigt wird. Unten über dem Fortschrittsbalken steht im Detail woran gerade gearbeitet wird. Dazu zeigt der Fortschrittsbalken den Fortschritt der aktuellen Aktion an.

6.3.2.3. Sprachauswahldialog

Der Sprachauswahldialog bietet beim Programmstart die Möglichkeit eine der in der Auswahlbox gegebenen Sprachen auszuwählen. Dieser Dialog wird nur angezeigt wenn durch die eingestellte Systemsprache keine Vorauswahl getroffen werden konnte.

6.3.3. Optionen

6.3.3.1. Start

Die Option „Start“ auf dem Anmeldebildschirm führt die Optionen des Bilder Generierens, Konvertierens zu XML und gepacktem XML, sowie des Hochladens zum weblin Server aus. Hierfür ist es erforderlich, dass alle Eingabefelder korrekt ausgefüllt sind. Alle drei anderen Optionen werden in der Reihenfolge „Erstelle Bilder“, „Erstelle XML“ und „Verschicke XML“ ausgeführt.

6.3.3.2. Erstelle Bilder

Um diese Option ausführen zu können, müssen Sie die Second Life Anmeldedaten angeben. Bis der Dialog das erste Mal reagiert, kann es unter Umständen ein paar Sekunden dauern, da im Hintergrund der Second Life Client geladen wird, der sich sofort versucht mit den angegebenen Daten beim Second Life Server anzumelden und zum Second Life Grid (agni) zu verbinden. Hierfür muss die „settings.xml“ im Anwendungsdaten-Ordner existieren, ansonsten wird das Programm beendet. Wenn die Verbindung erfolgreich ist, wird auf der rechten Seite ein Vorschaubild angezeigt. Sie müssen nun warten, bis Ihre Figur vollständig inklusive aller Texturen und Objekte geladen ist.

Notiz: Wenn andere Objekte, die nicht zu Ihrer Figur gehören auf dem Vorschaubild sichtbar sind und nicht nach drei Sekunden verschwunden sind, starten sie an dieser Stelle bitte das Programm neu. Falls der gleiche Fehler nach dem Start immer noch auftritt, müssen Sie ihre Figur an einer anderen Stelle in der Second Life Welt positionieren. Diese sollte möglichst leer sein.

Wählen Sie die Option „Start“ um das Rendern der Animationen zu Starten. Dazu werden die einzelnen Animationsdateien geladen, diese enden auf ".bvh" und beschreiben, wie die Figuren sich für die einzelnen Animationen bewegen müssen. Gegen Ende des Renderns der Animationssequenzen kann es sein, dass ihr Rechner langsam wird, da ca. 340 Megabyte Arbeitsspeicher für die Bilder benötigt werden. Zusätzlich werden ca. 140 Megabyte von Second Life benötigt. Im Folgenden werden die Bilder auf 100x100 Pixel optimiert und als PNG auf der Festplatte gespeichert. Wenn alle Punkte erfolgreich durchlaufen wurden, wird ein entsprechendes Nachrichtenfenster angezeigt, es sei denn es wurde auf dem

Anmeldedialog „Start“ ausgewählt. In diesem Fall wird mit dem nächsten Schritt, dem erstellen der XML Datei, fortgefahren

6.3.3.3. Erstelle XML

Um diese Option ausführen zu können müssen Sie keine Anmeldedaten angeben. Es ist auch keine aktive Verbindung zum Internet notwendig. Es wird hierfür die XML Datei mit den Gestiken geladen, sowie alle Bilder, die Sie im vorherigen Schritt generiert haben. Diese Bilder werden so konvertiert, dass sie in eine XML Datei eingefügt werden können. Diese Datei wird einmal als normale XML Datei[65] („send.xml“) und komprimierte Datei abgespeichert („send.xml.gz“). Wenn die gepackte Datei existiert wird diese im nächsten Schritt an den Server geschickt, da sie circa um ein Viertel kleiner ist. Sie sollten daher die gepackte Datei nicht löschen und auf Grund der Integrität der XML Datei weder diese, noch die gepackte bearbeiten. Wenn das Erstellen der Dateien erfolgreich war, wird ein entsprechendes Informationsfenster geöffnet, es sei denn im Anmeldedialog wurde die Option "Start" ausgewählt. In diesem Fall wird mit dem nächsten Schritt, dem Verschicken der XML Datei, fortgefahren.

6.3.3.4. Verschicke XML

Um diese Option ausführen zu können, müssen Sie die weblin Anmeldedaten angeben. Es muss ebenfalls eine generierte XML Datei existieren[66]. Diese wird vom weblin Second Life Generator geladen und mit den Zugangsdaten an den weblin Server geschickt. Dies kann unter Umständen etwas dauern, da die Datei größer als ein paar Kilobyte sein kann. Wenn das Hochladen erfolgreich abgeschlossen wurde, wird ein dementsprechendes Nachrichtenfenster angezeigt.

[65] Vgl. Kapitel 2.4.2.1
[66] Vgl. Kapitel 6.3.3.3

6.4. Fehlermeldungen

- **weblin Anmeldungsdaten unvollständig**

Meldung: „Bitte geben Sie Ihre weblin Anmeldungsdaten ein …“

Bedeutung: Sie haben die Anmeldungsdaten für Ihre Anmeldung bei weblin nicht vollständig angegeben.

Behebung: Tragen Sie Ihre Angaben vollständig in die dafür vorgesehenen Felder[67] ein.

- **Second Life Anmeldungsdaten unvollständig**

Meldung: „Bitte geben Sie Ihre Second Life Anmeldungsdaten ein …“

Bedeutung: Sie haben die Anmeldungsdaten für Ihre Anmeldung bei Second Life nicht vollständig angegeben.

Behebung: Tragen Sie Ihre Angaben vollständig in die dafür vorgesehenen Felder[68] ein.

- **Second Life Anmeldung fehlgeschlagen**

Meldung: „Anmeldung Fehlgeschlagen als [Vorname] [Nachname], falscher Benutzername oder Passwort“

Bedeutung: Das Anmelden am Second Life Anmeldeserver ist fehlgeschlagen. Entweder Benutzername oder Passwort sind falsch oder Sie wurden temporär gebannt.

Behebung: Geben Sie im Anmeldedialog Ihre richtigen Zugangsdaten für Second Life an. Wenn Sie kein Second Life Konto besitzen können Sie sich unter http://www.secondlife.com eines erstellen. Wenn Sie gebannt wurden versuchen Sie es später noch einmal.

- **Parsen der Animationsdefinitionsdatei fehlgeschlagen**

Meldung: „Parsen der Definitionsdatei für Animationen fehlgeschlagen“

Bedeutung: Die Datei „gestures.xml“ oder „gesturesFemale.xml“ im „WeblinAvatar“ Ordner existiert nicht oder ist fehlerhaft.

Behebung: Installieren Sie das Programm neu, wenn Sie nicht mit dem XML Format[69] vertraut sind.

---

[67] Vgl. Kapitel 6.3.2.1

[68] Vgl. Kapitel 6.3.2.1

[69] Vgl. Kapitel 2.4.1.3

- **Laden einer Animation fehlgeschlagen**

Meldung: „Laden der Animation fehlgeschlagen“

Bedeutung: Die Animationsdatei, welche auf „.bvh“ endet konnte nicht geladen werden.

Behebung: Wenn diese Datei nicht existiert, nicht lesbar oder fehlerhaft ist, müssen Sie entweder die Datei ersetzen, z.B. durch Neuinstallation, oder den entsprechenden Eintrag in der „gestures.xml“ bzw. „gesturesFemale.xml“ [70] korrigieren.

- **Speichern laufender Animationen**

Meldung: „Kann laufende Animationen nicht speichern“

Bedeutung: Während Animationen gerendert werden können diese nicht abgespeichert werden, da alle Animationen fertig gerendert sein müssen um optimiert zu werden.

Behebung: Starten Sie das Programm neu und versuchen Sie es erneut.

- **Ordner konnte nicht erstellt werden**

Meldung: „Erstellen des Ordners Fehlgeschlagen …“

Bedeutung: Der Ordner „WeblinAvatar“ existiert nicht und konnte auch nicht erstellt werden.

Behebung: Nur eine Neuinstallation kann diesen Fehler beheben, da einige wichtige Definitionsdateien nicht existieren. Überprüfen Sie nach der Installation die Zugriffsrechte.

- **Alte Bilder löschen**

Meldung: „Löschen von existierenden Dateien fehlgeschlagen – kann Dateien ohne Schreibrechte nicht speichern“

Bedeutung: Sie haben keine Schreibrechte für den „WeblinAvatar“ Ordner, die Unterordner und deren Dateien.

Behebung: Heben Sie den Schreibschutz für diesen Ordner und die Unterordner auf. Wenden Sie sich im Zweifelsfall an den Systemadministrator.

- **Datei lässt sich nicht öffnen**

Meldung: „Fehler beim Öffnen der Datei“

Bedeutung: Eine Bilddatei konnte nicht angelegt werden.

Behebung: Beenden Sie das Programm und überprüfen Sie Ihre Zugriffsrechte für den „WeblinAvatar“ Ordner und seine Unterordner.

[70] Vgl. Kapitel 2.4.1.3

- **Speichern fehlgeschlagen**

Meldung: „Fehler beim Speichern aufgetreten“

Bedeutung: Der Speichervorgang konnte nicht initialisiert werden.

Behebung: Starten Sie das Programm oder gegebenenfalls auch das Betriebssystem neu.

- **Lesbare Datei die nicht geöffnet werden kann**

Meldung: „Datei ist lesbar, kann aber nicht geöffnet werden“

Bedeutung: Ein Bild einer Animation konnte als lesbar geöffnet werden, kann aber nicht gelesen werden.

Behebung: Beenden Sie das Programm, überprüfen Sie ihre Zugriffsrechte und kontaktieren Sie im Zweifelsfall den Systemadministrator.

- **Daten werden schon geschickt**

Meldung: „Die Daten werden bereits an den weblin Server geschickt“

Bedeutung: Es existiert bereits ein HTTP Client der Daten an den weblin Server schickt oder geschickt hat.

Behebung: Starten Sie das Programm neu und führen Sie „Verschicke XML“[71] aus.

- **XML oder komprimiertes XML nicht gefunden**

Meldung: „Kann XML Datei oder komprimierte XML Datei nicht finden“

Bedeutung: Die Datei, welche zum weblin Server hochgeladen werden soll, existiert nicht.

Behebung: Falls Sie die Animationen bereits erfolgreich erstellt haben, führen Sie „Erstelle XML“ erneut aus. Andernfalls wählen Sie im Anmeldedialog die Option „Start“.

- **Zu kleine XML Datei**

Meldung: „Ungültige XML Datei. Die Datei ist zu klein.“

Bedeutung: Die XML Datei ist ungültig, da Ihre Größe nicht ausreicht um Animationen darzustellen.

Behebung: Generieren Sie die Bilder und die XML Datei neu. Wenn dies nicht zur Behebung des Fehlers führt bearbeiten Sie, bei ausreichender Kenntnis in XML, die Dateien „gestures.xml“ und „gesturesFemale.xml“. Ansonsten installieren Sie bitte „wSLGen“ neu.

[71] Vgl. Kapitel 6.3.3.4

- **Falsche weblin Anmeldedaten**

Meldung: „Falscher weblin Benutzername oder Passwort“

Bedeutung: Der von Ihnen angegebene Benutzername oder das Passwort ist falsch.

Behebung: Geben Sie im Anmeldedialog Ihr richtigen Benutzernamen und das richtige Passwort an. Wenn das Generieren der XML Datei bereits abgeschlossen wurde wählen Sie danach nur „Verschicke XML“[72] aus. Falls Sie kein weblin Konto besitzen können Sie auf der weblin Webseite[73] eines erstellen.

- **HTTP Fehler**

Meldung: „HTTP Fehler …“

Bedeutung: Beim Senden der Animationen in der XML Datei ist ein HTTP Fehler aufgetreten. Der zugehörige Statuscode ist mit angegeben.

Behebung: Versichern Sie sich, dass Ihre Internetverbindung aktiv ist und Sie eine Verbindung zur Seite von weblin aufbauen können[74], sowie dass Sie keine Firewall aktiv haben, die „wSLGen“ blockt.

- **Fehlen der „settings.xml“**

Meldung: „Second Life Error: ERROR: Invalid name for LLDir::setLindenUserDir“

Bedeutung: Die Datei „settings.xml“ ist weder im „WeblinAvatar“ Ordner noch in „Anwendungsdaten\wSLGen\user_settings“ zu finden oder beschädigt.

Behebung: Bitte installieren Sie „wSLGen“ neu oder kopieren Sie aus demselben Ordner des „Second Life“ Verzeichnisses die „settings.xml“ Datei.

- **Andere Second Life Fehler**

Meldung: „Second Life Fehler: …“

Bedeutung: Dies ist ein Fehler, der von dem Second Life Client ausgelöst wurde.

Behebung: Um diesen Fehler zu beheben wenden Sie sich bitte an Linden Labs, vorausgesetzt er wurde hier nicht weiter beschrieben.

---

[72] Vgl. Kapitel 6.3.3.4

[73] Link zur Herstellerseite: http://www.weblin.com/

[74] Link zur Herstellerseite: http://www.weblin.com/

6.5. Wiederanlaufbedingungen

Nach einem Absturz oder Beendigung des Programms durch einen Fehler reicht es im Normalfall aus, das Programm wieder zu starten[75]. Wenn dabei Dateien beschädigt wurden und der normale Programmstart nicht funktionieren sollte, versuchen Sie das Programm zu deinstallieren und erneut zu installieren. Nur in diesem Fall kann die Integrität der Daten gewährleistet werden. Dabei gilt immer die Voraussetzung, dass Sie die nötigen Schreibrechte auf Ihrem System besitzen und keine benötigten Ordner gegen das Beschreiben geschützt sind.

[75] Vgl. Kapitel 6.3.1

Literaturverzeichnis

Wolf, Heiner, 2005: Virtual Presence, Internet http://virtual-presence.org/, Stand 2008-02-15, Abruf 2008-02-15

Schnittstellendokumentation

Für die Schnittstellendokumentation sind zwei Referenzen vorhanden. Die komplette Referenz und eine stark gekürzte Version, die nur komplett neu erstellte Dateien des „weblin Second Life Generator“ enthält, nicht aber veränderte Dateien des Second Life Clients.

- Komplettreferenz: Die Komplettreferenz ist im auf der beiliegenden CD befindlichen Ordner „Schnittstellendokumentation“ mit der „index.html“ Datei abzurufen und umfasst den kompletten Quellenordner des modifizierten Second Life Programms.
- Kurzreferenz: Die Kurzreferenz ist im auf der beiliegenden CD befindlichen Ordner „Schnittstellendokumentation“ mit der „index_short.html“ Datei abzurufen und umfasst ausschließlich die neu erstellten Dateien.